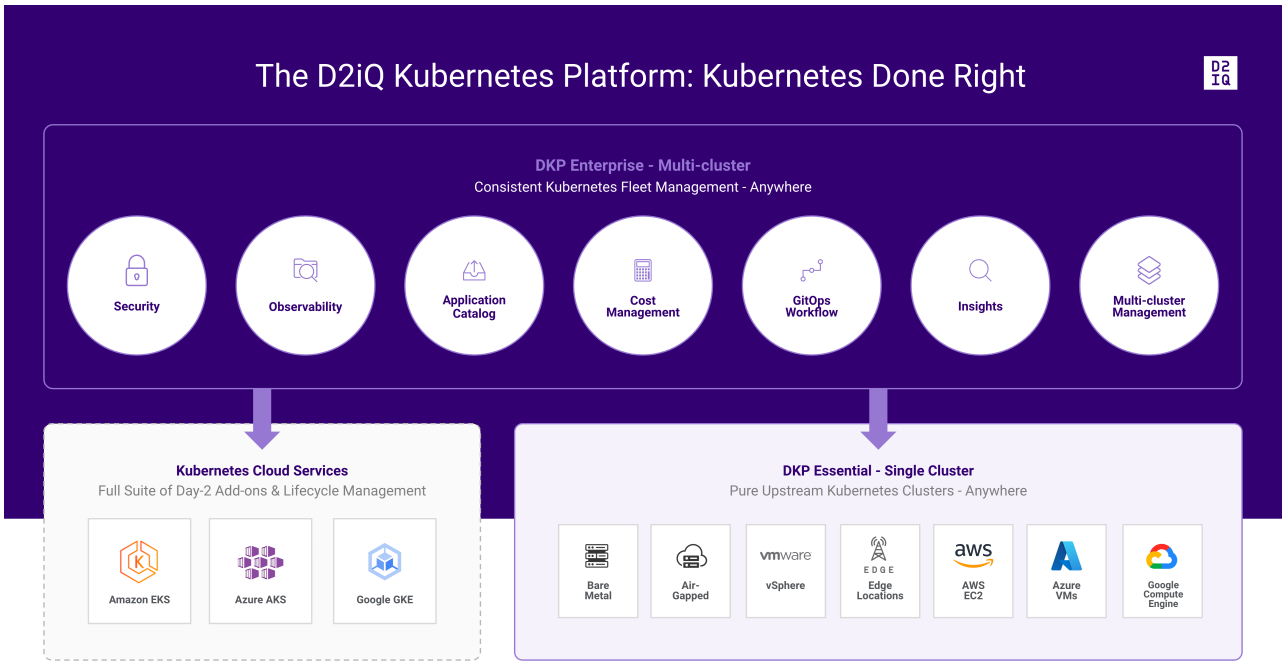
Introducing the D2iQ Kubernetes Platform (DKP)

D2iQ Kubernetes Platform (DKP)

Exported on 06/06/2024

Table of Contents

As the leading independent Kubernetes Management Platform in production, the D2iQ Kubernetes Platform (DKP) provides a holistic approach and a complete set of enterprise-grade technologies, services, training, and support to build and run applications in production at scale. Built around the open-source Cluster API, the new version of DKP becomes the single, centralized point of control for an organization's application infrastructure, empowering organizations to more easily deploy, manage, and scale Kubernetes workloads in Day 2 production environments.



Features	Benefits
Container Orchestration	Leverage an industry standard distribution of open-source Kubernetes for cluster and container management.
Application Management and Deployment	Deploy applications and services within Kubernetes clusters with Helm[1].
Observability	Gain deep insight into your Kubernetes clusters and applications with open source metrics leveraging Telegraf, Prometheus, and Grafana[2].

1 https://helm.sh/
2 https://grafana.com/docs/

Features	Benefits
Cluster API	Automate cluster lifecycle management using Cluster API (see page 82) to simplify the provisioning, upgrading, and operating many Kubernetes clusters across a wide range of distributions and virtual and physical environments.
Cluster Autoscaling	Save operational costs by automatically scaling down (see page 1241) capacity when it's not needed and adding capability when there is greater demand, with CAPI enabled autoscaling groups.
Logging	Collect and analyze logs and metrics (see page 928) to ensure optimal performance and troubleshooting with Grafana (see page 1015), Loki, and Fluentbit (see page 952).
Cloud-Native Scale Testing	Extensive integration and workload testing at massive scale with a wide range of workloads to ensure real-world preparedness.
Networking and Routing	Easily automate and expose application endpoints (see page 984) with Calico, Traefik (see page 976), and CoreDNS.
Fine-Grained Cluster	Upgrades (see page 1705) reduce operational overhead with non-disruptive patching or parallel worker node upgrades.
Backup and Recovery	Ensure business continuity and disaster recovery (see page 904) with Velero[3].
Declarative Automated Installer With Day 2 Platform Services	Accelerate time-to-production on any infrastructure (see page 146) with consistency and reliability with the required platform applications (see page 662) needed for Day 2 production (see page 590).
Operate in Air-gapped Environments (see page 312)	Leverage declarative APIs to optimize cluster resources for cost, resilience, and performance.

3 https://velero.io/docs/main/

Features	Benefits
End-to-End Support	Enterprise-grade support[4] and services for both Kubernetes and its supported platform applications.

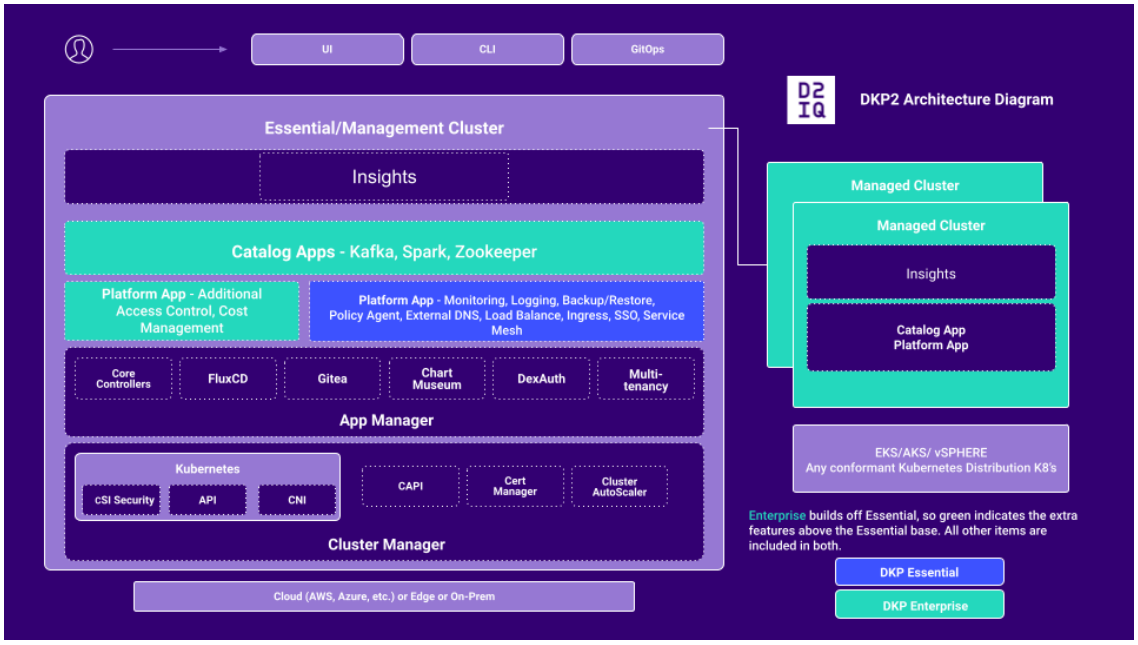
4 https://support.d2iq.com/hc/en-us

1 Architecture

1.1 Learn the key concepts and architectural components of a DKP cluster

Kubernetes creates the foundation for a DKP cluster. Because of this fundamental relationship, you should be familiar with a few key concepts and terms. The topic in this section gives a brief overview of the native Kubernetes architecture, a simplified view of the DKP architecture - both Essential and Enterprise versions. Also, it shows the operational workflow for a DKP cluster.

The following diagram is an architectural overview to help you visualize the flow of the key components:



1 DKP Architectural overview

1.2 Components for the Kubernetes Control Plane

The native Kubernetes cluster consists of **components** in the cluster's **control plane** and **worker nodes** that run containers and maintain the runtime environment.

DKP supplements the native Kubernetes cluster by including a pre-defined and pre-configured set of applications. Because this pre-defined set of applications provides critical features for managing a Kubernetes cluster in a production environment, the default set is identified as DKP **platform applications**.

See Platform applications (see page 590) for the full set of DKP platform services.

1.3 Related Information

For information on related topics or procedures, see the Kubernetes[5] documentation.

1.4 Next Steps

- DKP Ports (see page 65)
- Supported Infrastructure Operating Systems (see page 67)

1.5 DKP Ports

This section describes ports used by the different Kubernetes components in your DKP cluster. For more information regarding ports, refer to the Kubernetes Documentation on this page: Ports and Protocols[6]

1.5.1 Control-plane nodes

Port	DKP Component	Notes
22		ssh
179	calico-node	BGP
1338	Containerd	metrics
2379	etcd	client
2380	etcd	peer
6443	kube-apiserver	
9091	calico-node	felix metrics
9092	calico-node	bird metrics

5 https://kubernetes.io/docs/concepts/overview/components/
6 https://kubernetes.io/docs/reference/networking/ports-and-protocols/

Port	DKP Component	Notes
9099	calico-node	felix liveliness
9100	prometheus node-exporter	metrics
10248	kubelet	health
10249	kube-proxy	metrics
10250	kubelet	
10256	kube-proxy	health
10257	kube-controller-manager	secure port
10259	kube-scheduler	secure port
30000-32767	Kubernetes NodePorts	

1.5.2 Worker nodes

Port	DKP Component	Notes
22	Ansible	ssh
179	calico-node	BGP
1338	Containerd	metrics
5473	calico-typha	syncserver
9091	calico-node	felix metrics
9092	calico-node	bird metrics

Port	DKP Component	Notes
9093	calico-typha	metrics
9099	calico-node	felix liveliness
9100	prometheus node-exporter	metrics
9400	NVIDIA GPU DCGM	metrics
10248	kubelet	health
10249	kube-proxy	metrics
10250	kubelet	
10256	kube-proxy	health
30000-32767	Kubernetes NodePorts	

1.5.3 Next Topic:

Supported Infrastructure Operating Systems (see page 67)

1.6 Supported Infrastructure Operating Systems

Below are listed the tested and supported Operating Systems for DKP:

> If the full table is not showing below, select the box icon in the upper-right corner to expand the view:

1.6.1 Amazon Web Services (AWS)

Operating System	Kernel	Default Config (see page 295)	FIPS (see page 333)	Air Gapped (see page 312)	FIPS with Air Gapped (see page 350)	GPU Support (see page 369)	GPU with Air Gapped (see page 389)	Konvoy Image Builder (see page 1626)
CentOS 7.9[7]	3.10.0-1160.80.1.el7.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 7.9[8]	3.10.0-1160.80.1.el7.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 8.4[9]	4.18.0-305.88.1.el8_4.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 8.6[10]	4.18.0-372.70.1.el8_6.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 8.8[11]	4.18.0-477.27.1.el8_8.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Ubuntu 18.04 (Bionic Beaver)[12]	5.4.0-1103-aws	Yes						Yes
Ubuntu 20.04 (Focal Fossa)[13]	5.15.0-1051-aws	Yes				Yes		Yes

7 https://wiki.centos.org/Manuals(2f)ReleaseNotes(2f)CentOS7(2e)2009.html
8 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/7.9_release_notes/index
9 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.4_release_notes/index
10 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/8.6_release_notes/index
11 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.8_release_notes/index
12 https://wiki.ubuntu.com/BionicBeaver/ReleaseNotes
13 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes

Operating System	Kernel	Default Config (see page 295)	FIPS (see page 333)	Air Gapped (see page 312)	FIPS with Air Gapped (see page 350)	GPU Support (see page 369)	GPU with Air Gapped (see page 389)	Konvoy Image Builder (see page 1626)
SLES 15 SP3[14]	5.14.21-150500.55.7-default	Yes				Yes		Yes
Oracle Linux RHCK 7.9[15]	3.10.0-1160.105.1.0.1.el7.x86_64	Yes	Yes					Yes
Rocky Linux 9.1[16]	5.14.0-162.12.1.el9_1.0.2.x86_64	Yes		Yes				Yes
Flatcar 3033.3.x[17]	5.10.198-flatcar	Yes						Yes

1.6.2 Microsoft Azure

Operating System	Kernel	Default Config (see page 541)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1642)
Ubuntu 20.04 (Focal Fossa)[18]	5.15.0-1053-azure	Yes						Yes

14 https://documentation.suse.com/en-us/sles/15-SP3/
15 https://docs.oracle.com/en/operating-systems/oracle-linux/7/relnotes7.9/
16 https://docs.rockylinux.org/release_notes/9_1/
17 https://www.flatcar.org/releases/#lts-release
18 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes

Operating System	Kernel	Default Config (see page 541)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1642)
Rocky Linux 9.1[19]	5.14.0-162.12.1.el9_1.0.2.x86_64	Yes						Yes

1.6.3 Google Cloud Platform (GCP)

Operating System	Kernel	Default Config (see page 572)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1646)
CentOS 7.9[20]	3.10.0-1160.62.1.el7.x86_64	Yes						Yes
Ubuntu 18.04[21]	5.4.0-1072-gcp	Yes						Yes
Ubuntu 20.04[22]	5.13.0-1024-gcp	Yes						Yes

1.6.4 Pre-Provisioned

Pre-provisioned (see page 148) includes on-premises, vSphere, AWS, Azure, and GCP infrastructures and is described in more detail under What is Pre-provisioned Infrastructure (see page 84).

19 https://docs.rockylinux.org/release_notes/9_0/
20 https://wiki.centos.org/action/show/Manuals/ReleaseNotes/CentOS7.2003
21 https://wiki.ubuntu.com/BionicBeaver/ReleaseNotes
22 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes

Operating System	**Kernel**	Default Config (see page 152)	FIPS (see page 194)	Air Gapped (see page 169)	FIPS with Air-Gapped (see page 214)	GPU Support (see page 239)	GPU with Air Gapped (see page 262)	Konvoy Image Builder (see page 1660)
CentOS 7.9[23]	3.10.0-1160.80.1.el7.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 7.9[24]	3.10.0-1160.80.1.el7.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 8.4[25]	4.18.0-305.88.1.el8_4.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 8.6[26]	4.18.0-372.70.1.el8_6.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 8.8[27]	4.18.0-477.27.1.el8_8.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Flatcar 3033.3.x[28]	3033.3.16-flatcar	Yes						Yes
Ubuntu 20.04[29]	5.15.0-1048-aws	Yes				Yes		Yes
SLES 15 SP3[30]	5.3.18-59.37-default	Yes				Yes		Yes

23 https://wiki.centos.org/action/show/Manuals/ReleaseNotes/CentOS7.2003
24 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/7.9_release_notes/index
25 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.4_release_notes/index
26 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/8.6_release_notes/index
27 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.8_release_notes/index
28 https://www.flatcar.org/releases/#lts-release
29 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes
30 https://documentation.suse.com/en-us/sles/15-SP3/

Operating System	Kernel	Default Config (see page 149)	FIPS (see page 194)	Air Gapped (see page 169)	FIPS with Air-Gapped (see page 214)	GPU Support (see page 239)	GPU with Air Gapped (see page 262)	Konvoy Image Builder (see page 1660)
Oracle Linux RHCK 7.9[31]	5.4.17-2136.314.6.3.el7uek.x86_64	Yes						Yes
Rocky Linux 9.1[32]	5.14.0-162.12.1.el9_1.0.2.x86_64	Yes		Yes				Yes

1.6.5 Pre-provisioned/Azure

Operating System	Kernel	Default Config[33]	FIPS[34]	Air Gapped[35]	FIPS with Air Gapped[36]	GPU Support[37]	GPU with Air Gapped[38]	Konvoy Image Builder[39]
RHEL 8.6[40]	4.18.0-372.52.1.el8_6.x86_64	Yes	Yes	Yes	Yes			Yes

1.6.6 vSphere

vCenter Server + ESXi

31 https://docs.oracle.com/en/operating-systems/oracle-linux/7/relnotes7.9/
32 https://docs.rockylinux.org/release_notes/9_1/
33 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273320894/2.6%2BPre-provisioned%2BNon-air-gapped%2BInstall
34 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273385679/2.6%2BPre-provisioned%2BFIPS%2BInstall
35 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273321203/2.6%2BPre-provisioned%2BAir-gapped%2BInstall
36 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273647809/2.6+Pre-provisioned+FIPS+Air-gapped+Install
37 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273387183/2.6+Pre-provisioned+with+GPU+Install
38 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273319903/2.6+Pre-provisioned+Air-gapped+with+GPU+Install
39 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273713682/2.6+KIB+with+Pre-provisioned+Environments
40 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/8.6_release_notes/index

Operating System	Kernel	Default Config (see page 447)	FIPS (see page 492)	Air Gapped (see page 468)	FIPS with Air Gapped (see page 514)	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1652)
RHEL 7.9[41]	3.10.0-1160.el7.x86_64	Yes	Yes	Yes	Yes			Yes
RHEL 8.4[42]	4.18.0-305.el8.x86_64	Yes	Yes	Yes	Yes			Yes
RHEL 8.6[43]	4.18.0-372.9.1.el8.x86_64	Yes	Yes	Yes	Yes			Yes
RHEL 8.8[44]	4.18.0-477.10.1.el8_8.x86_64	Yes	Yes	Yes	Yes			Yes
Ubuntu 20.04[45]	5.4.0-125-generic	Yes		Yes				Yes
Rocky Linux 9.1[46]	5.14.0-162.6.1.el9_1.x86_64	Yes		Yes				Yes
Flatcar 3033.3.x[47]	3033.3.16-flatcar	Yes						Yes

41 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/7.9_release_notes/index
42 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.4_release_notes/index
43 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/8.6_release_notes/index
44 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.8_release_notes/index
45 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes
46 https://docs.rockylinux.org/release_notes/9_1/
47 https://www.flatcar.org/releases/#lts-release

1.6.7 VMware Cloud Director (VCD)

Operating System	Kernel	Default Config (see page 1444)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1652)
Ubuntu 20.04[48]	5.4.0-125-generic	Yes						Yes

1.6.8 Amazon Elastic Kubernetes (EKS)

Operating System	Kernel	Default Config (see page 413)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1642)
Amazon Linux 2 v7.9[49]	5.10.199-190.747.amzn2.x86_64	Yes						

48 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes
49 https://docs.aws.amazon.com/AL2/latest/relnotes/relnotes-al2.html

1.6.9 Azure Kubernetes Service (AKS)

Operating System	Kernel	Default Config (see page 556)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1646)
Ubuntu 22.04.2 LTS[50]	5.15.0-1051-azure	Yes						

1.6.10 Next Step:

Download DKP (see page 76)

50 https://discourse.ubuntu.com/t/jammy-jellyfish-release-notes/24668

2 Download DKP

To download a new version of DKP, you have 2 options:

2.1 Download from the Support Website

Download DKP

From the Download site, select the DKP binary for either Darwin(MacOS) or Linux. After download, extract for your system.

To extract a tar file, either:

- MacOS/Darwin - Use a file manager program like 7-Zip and right-click the tar file. Select “Open With” and then 7-Zip File Manager. Select “Extract” and choose a location to save the extracted files.
- Linux - Use the command line and type `tar -xvf filename.tar` to extract the files to the current directory. You can add tar `-xvzf` option if the file is compressed with gzip.

2.1.1 AWS Marketplace Only

Follow the instructions on the AWS console to download the container image.

After downloading the image, run this command to copy the binaries to your host:

```
docker run -it --rm -u $(id -u):$(id -g) -v $(pwd):/dkp $CONTAINER_IMAGES
```

A successful operation returns this output:

```
dkp binary is placed in the local directory, to run:
./dkp --help
```

And you can view the `dkp` binary in your working directory. Follow the DKP installation (see page 127) instructions using these binaries, and then add your license (see page 98) to DKP. If you have problems downloading or installing DKP, contact your sales representative or sales@d2iq.com[51].

2.1.2 Next Step:

Day 0 - Get Started with DKP (see page 77)

51 mailto:sales@d2iq.com

3 Day 0 - Get Started with DKP

At D2iQ, we partner with you throughout the entire cloud-native journey, at the beginning on Day 0 all the way through your Day 2 operations. Here is the breakdown:

- **Day 0** (see page 77) is the planning phase where we introduce definitions and concepts.
- **Day 1** (see page 146) is to install the software and get it up and running.
- **Day 2** (see page 590) involves customizations, application, and operations management. In the end, we provide ongoing maintenance, keep the platform resilient, keep it secure, and make sure it can be upgraded.

When installing DKP, first determine the infrastructure on which you want to deploy. Then, ensure you understand the basic concepts and have the prerequisites met, like storage and resource requirements, using the information contained in this section of the documentation.

For example, you can:

- Install on a public cloud infrastructure, such as Amazon Web Services (AWS), GCP, or Azure.
- Install on an internal network, on-premises environment, with a physical or virtual infrastructure.
- Install on an Air-gapped environment.
- Install with or without FIPS and GPU.

After you choose your environment, you download DKP (see page 76), and then select the Day 1 - Basic Install instructions (see page 146) for your Infrastructure provider and environment. The basic installs set up a cluster with the Konvoy component, and then install the Kommander component, so that you can access the dashboards through the DKP UI. You can use these basic clusters to explore DKP and prepare to take your clusters into *production*, where you will deploy and enable the applications that support Day 2 operations. Then you can deploy and test your workloads, and put Kubernetes to work!

After you finish the basic install and are ready to customize, move to Custom Installation and Additional Infrastructure Tools (see page 1050), if desired. Otherwise, proceed to Day 2 Cluster Operations Management (see page 590) to make the software exactly what you want it to be.

Follow this list to ensure you have the resource requirements met for a successful implementation:

- DKP Concepts and Terms (see page 78)
- Licenses (see page 85)
- Provide Context for Commands with a kubeconfig File (see page 106)
- Storage (see page 108)
- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

3.1 DKP Concepts and Terms

Understand the terminology used in association with DKP with the information on this page - DKP concepts and terms.

DKP is composed of three main components, Konvoy, Kommander, and Konvoy Image Builder (KIB) that work together to provide a single and centralized point of control for an organization's application infrastructure. DKP empowers organizations to deploy, manage, and scale Kubernetes workloads in Day 2 production environments more easily.

Each main component specifically manages the following:

- **Konvoy** is the cluster lifecycle manager component of DKP. Konvoy relies on Cluster API, Calico, and other open-source and proprietary software to provide simple cluster lifecycle management for conformant Kubernetes clusters with networking and storage capabilities.
  Konvoy uses industry-standard tools to provision certified Kubernetes clusters on multiple cloud providers, vSphere, and on-premises hardware in connected and air-gapped environments. Konvoy contains the following components:
    - **Cluster Manager** - Cluster API, CSI, CNI, Cluster AutoScaler, Cert Manager, MetalLB

The Konvoy component is installed according to the **cluster's infrastructure**.

- To install DKP quickly and without much customization, refer to the Day 1 - Basic Installs by Infrastructure (see page 146) section.
- To choose more environment and cluster customizations, refer to theCustom Installation and Additional Infrastructure Tools (see page 1050) section for more information.

- **Kommander** is the fleet management component of DKP. Kommander delivers centralized observability, control, governance, unified policy, and better operational insights. With DKP Essential, Kommander manages a single Kubernetes cluster.
  In DKP Enterprise, Kommander supports attaching workload clusters and lifecycle management of clusters using Cluster API. DKP Enterprise also offers lifecycle management of applications through FluxCD. Kommander contains the following components:
    - User interface, Security, Observability, Networking, Application Management
    - *Platform Applications*: Applications that come out of the box with DKP providing functionality like observability, cost management, monitoring, logging, making DKP clusters Day 2 production ready right from install. Platform applications are D2iQ's choice of selected applications from the open source community that are consumed by the platform.
    - **Essential Platform Applications:** Monitoring, Logging, Backup/Restore, Policy Agent, External DNS, Load Balance, Ingress, SSO, Service Mesh.
    - **Enterprise Platform Applications:** All above Essential Platform Applications, plus additional Access Control and Centralized Cost Management.
    - *Catalog Applications*: Applications in DKP Enterprise that are deployed to be used for customer workloads, such as Kafka, Spark and ZooKeeper.

The Kommander component is installed according to the **cluster's environment type**. See Installing Kommander by Environment (see page 1530) for more information.

- **Konvoy Image Builder** (KIB) creates Cluster API-compliant machine images. It configures those images to contain all the necessary software to deploy Kubernetes cluster nodes. See Konvoy Image Builder (see page 1626) for more information.

DKP Essential and Enterprise also provide a helpful add-on called DKP Insights[52].

3.1.1 Related Topics:

Universal Configurations for all Infrastructure Providers (see page 1052)

3.1.2 Next Topic:

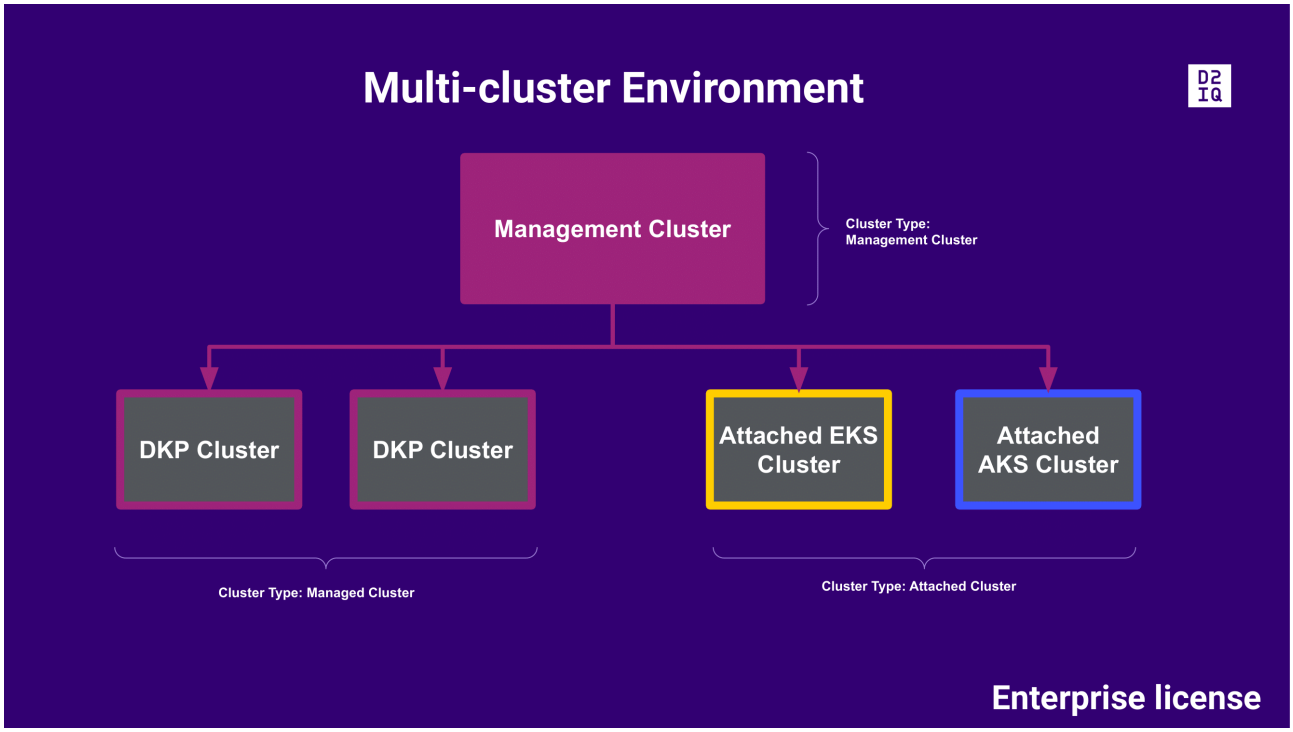
What is Pre-provisioned Infrastructure (see page 84)

3.1.3 Cluster Types and Concepts

Cluster types such as Management clusters, Managed clusters, and Attached clusters are key concepts in understanding and getting the most out of DKP Essential versus Enterprise environments.

52 https://d2iq.atlassian.net/wiki/spaces/DINS

3.1.3.1 Multi-cluster Environment



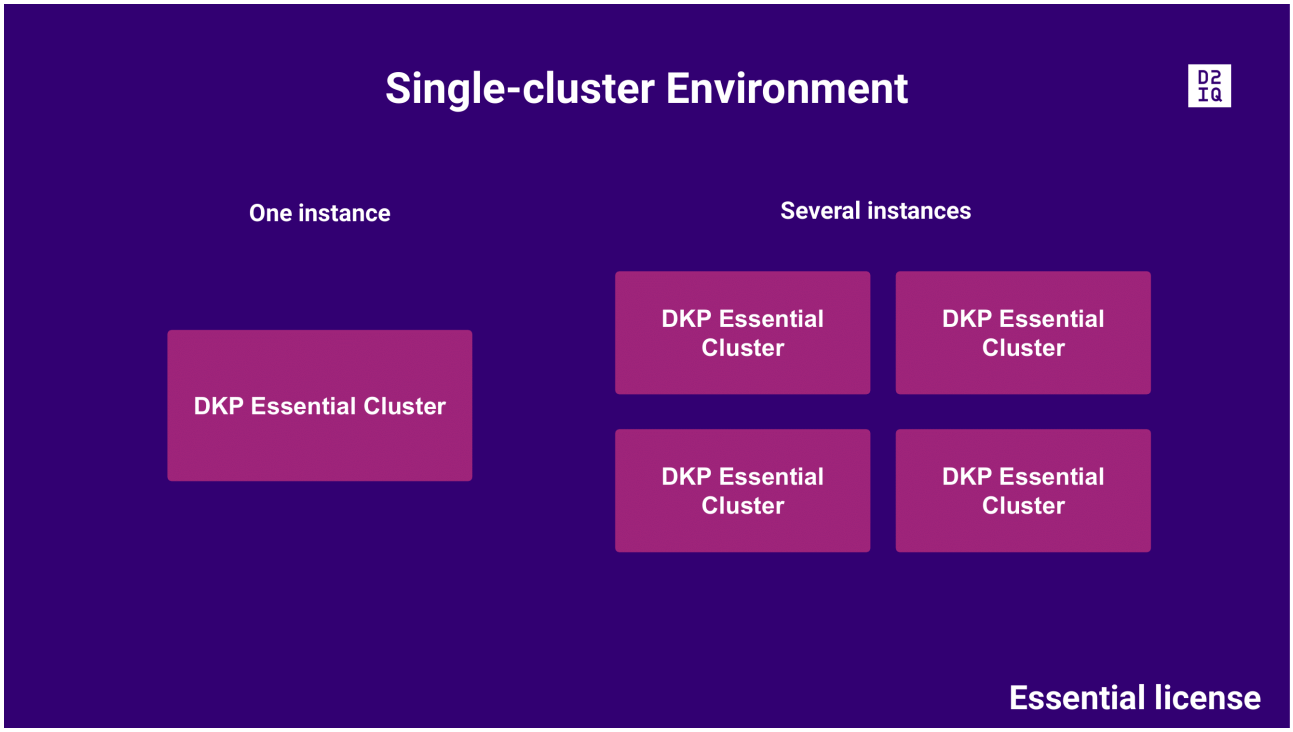
Management Cluster: Is the cluster where you install DKP, and it is self-managed. In a multi-cluster environment, the Management cluster also manages other clusters.

Customers with an Enterprise license (see page 89) should run workloads on *Managed* and *Attached* clusters, and not on the *Management cluster*.

Managed Cluster: Also called a "DKP cluster," this is a type of workload cluster that you can create with DKP. The *DKP Management cluster* manages its infrastructure, its lifecycle, and its applications.

Attached Cluster: A type of workload cluster that is created outside of DKP, but is then connected to the *DKP Management Cluster* so that it can be managed by DKP. In these cases, the DKP Management cluster only manages the attached cluster's applications.

3.1.3.2 Single-cluster Environment



DKP Essential Cluster: Is the cluster where you install DKP. A DKP Essential cluster is a stand-alone cluster. It is self-managed and therefore capable of provisioning itself. In this single-cluster environment, you cannot attach other clusters and all workloads are run on your *DKP Essential cluster*. You can, however, have several separate DKP Essential instances, each with its own license.

Customers with an Essential license (see page 91) can run workloads on their *DKP Essential cluster*.

> If you have not decided which license to get, but plan on adding one or several clusters to your environment, and manage them centrally, D2iQ recommends obtaining an Enterprise license (see page 89).

3.1.3.3 Other important concepts

Self-managed Cluster: In a DKP landscape, only *DKP Essential* and *DKP Enterprise Management* clusters are self-managed. *Self-managed clusters* are clusters that manage the provisioning, and deployment of its own nodes through CAPI controllers. The CAPI controllers are a managing entity, which automatically manages the lifecycle of a cluster's nodes based on a customizable definition of the resources.

A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing. As part of the underlying processing using the `--self-managed` flag, the DKP CLI:

- creates a bootstrap cluster
- creates a workload cluster
- moves CAPI controllers from the bootstrap cluster to the workload cluster, making it self-managed
- deletes the bootstrap cluster

Network-Restricted Cluster:

(NOT equivalent to air-gapped)

A network-restricted or firewalled cluster is secured by a firewall, DMZ, NAT gateway, proxy server, or requires additional access information. Network-restricted clusters are usually located in remote locations or at edge, and therefore not in the same network as the Management cluster.

The main difference between *network-restricted* and *air-gapped* clusters is that network-restricted clusters can reach external networks (like the Internet), but its services or ingresses cannot be accessed from outside. Air-gapped clusters, on the other hand, do not allow ingress, nor egress traffic.

In a multi-cluster environment, DKP supports attaching a network-restricted cluster (see page 804) to a DKP Management cluster. You can also enable a proxied access (see page 846) pipeline through the Management cluster, which allows you to access the network-restricted cluster's dashboards without being in the same network.

3.1.3.4 Next Topic:

Provide Context for Commands with a kubeconfig File (see page 106)

3.1.4 CAPI Concepts and Terms

DKP uses CAPI (ClusterAPI) technology for creating and managing the lifecycle of Kubernetes Clusters. A basic understanding of CAPI concepts and terms is helpful in understanding how to install and maintain DKP. You can find a deeper discussion of the architecture in the ClusterAPI Book[53].

CAPI makes use of a bootstrap cluster for provisioning and starting clusters. A bootstrap cluster handles the following actions:

1. Generating the cluster certificates, if they are not otherwise specified.
2. Initializing the control plane, and managing the creation of other nodes until it is complete.
3. Joining control plane and worker nodes to the cluster.
4. Installing and configuring networking plugin (Calico CNI), CSI volume provisioners, cluster-autoscaler and other core Kubernetes components.

BootstrapData is machine or node role-specific data, such as cloud initialization data, used to bootstrap a "machine" onto a node.

53 https://cluster-api.sigs.k8s.io/user/concepts.html

For customers using Kommander for multi-cluster management, a **management cluster** manages the lifecycle of workload clusters. As the management cluster, DKP Kommander works with bootstrap providers, infrastructure providers, and maintains cluster resources such as bootstrap configurations and templates. If you are working with only one cluster, Kommander will provide you with addon (platform application) management for that cluster, but not others.

A **workload cluster** is a Kubernetes cluster whose lifecycle is managed by a management cluster, and provides the platform on which you deploy and execute your workloads.

As part of a collection of **Custom Resource Definitions** or **CRDs** that extend the Kubernetes API, these additional concepts are important for understanding the upgrade.

A **ClusterResourceSet** Kubernetes cluster created by CAPI is functionally minimal in nature. Crucial components like CSI and CNI are not a part of the default cluster spec. A ClusterResourceSet is a CRD that can be used to group and deploy core cluster components post-Kubernetes cluster install.

When you create a bootstrap cluster. You can find all the components in the default namespace and we move them to the workload cluster during the process of making the cluster self-managed.

A **machine** is a declarative spec for a platform or infrastructure component that hosts a Kubernetes node such as a bare metal server or a VM. CAPI uses provider-specific controllers to provision and install new hosts which then register as nodes. When you update a machine spec other than for certain values, such as annotations, status, and labels, the controller deletes the host and creates a new one that conforms to the new spec. This is called machine immutability. If you delete a machine, the controller deletes both the infrastructure and the node. Provider-specific information is not portable between providers.

Within CAPI, you use declarative **MachineDeployments** to handle changes to machines by replacing them in much the same way that a core Kubernetes Deployment replaces Pods. MachineDeployments reconcile changes to machine specs by rolling out changes to two **MachineSets** (similar to a ReplicaSet), both the old and the newly-updated.

A **MachineHealthCheck** identifies unhealthy node conditions, and initiates remediation for nodes owned by a MachineSet.

3.1.4.1 Related Topics:

Customizing CAPI Components for a Cluster (see page 1061)

3.1.4.2 Next Topic:

What is Pre-provisioned Infrastructure (see page 84)

3.1.5 Air-gapped or Non-air-gapped Environment?

3.1.5.1 Air-gapped Environments

In an air-gapped environment, your environment is **isolated** from unsecured networks, like the Internet. Running your workloads on an air-gapped environment is common in scenarios where **security** is a determining factor. DKP in air-gapped environments allows you to manage your clusters while shielding it from undesired external influences.

You can create an air-gapped cluster on on-premise environments or any supported cloud infrastructure (see page 67). In this configuration, you are responsible for providing an image registry. You must also retrieve required artifacts and configure DKP to use those from a local directory when creating and managing DKP clusters.

Secure interactions with other networks

There are a number of ways you can perform actions that require incoming data from other networks in spite of your environment's isolation: in some configurations, air-gapped clusters allow inbound connections, but cannot initiate outbound connections. In other configurations, you can set up a bastion host, which serves as a gateway between the Internet (or other untrusted networks) and your environment, and facilitates the download of install, upgrade, etc. files and images that are required for other machines to run in air-gapped environments.

Common industry synonyms:

Fettered, disconnected, restricted, SIPR, etc.

3.1.5.2 Non-air-gapped Environments

In a non-air-gapped environment, your environment has two-way access to and from the **Internet**. You can create a non-air-gapped cluster on pre-provisioned (on-premise) environments or any cloud infrastructure.

DKP in a non-air-gapped environment allows you to manage your clusters while facilitating **connections** and offering integration with other tools and systems.

Common industry synonyms:
Open, accessible (to the Internet), not restricted, NIPR, etc.

3.1.5.3 Next Step:

What is Pre-provisioned Infrastructure (see page 84)

3.1.6 What is Pre-provisioned Infrastructure

Pre-provisioned as an infrastructure type allows the deployment of Kubernetes using DKP® to pre-existing machines. Other providers such as vSphere®, AWS®, or Azure® create, or "provision," the machines themselves before Kubernetes is deployed. On most infrastructures (including vSphere and cloud providers), DKP provisions the actual nodes automatically as part of deploying a cluster. It creates the VMs using the appropriate image, then handles the networking and installation of Kubernetes.

However, DKP can also work with pre-provisioned infrastructure in which *you* provision the VMs for the nodes themselves. Note that you can pre-provision nodes for DKP on bare metal, on vSphere, or even in the cloud. Pre-provisioned and vSphere are a combination of the physical (on-premises bare metal) and virtual servers (VMware® vSphere).

3.1.6.1 Where is Pre-provisioned used?

Pre-provisioned environments are often used in bare metal deployments, where you deploy your OS (see page 67) (such as RHEL® or Ubuntu™, and so on) on physical machines. Creating a pre-provisioned cluster means that you, as an **Infrastructure Ops Manager**, are responsible for allocating your compute resources, setting up networking, collecting IP and SSH information to be provided to DKP, etc. You can then provide all of th

needed details to the pre-provisioned provider to deploy Kubernetes. These operations are done manually or with the help of other tools.

In pre-provisioned environments, DKP handles your cluster's lifecycle (installation, upgrade, node management, and so on). DKP installs Kubernetes, monitoring and logging apps, as well as its own user interface.

The main use cases for the pre-provisioned provider are:

- On-premises clusters
- Cloud or IAAS environments that do not currently have a D2iQ® supported infrastructure provider
- Cloud environments where you must use predefined infrastructure instead of having one of the supported cloud providers create it for you

In an environment with access to the Internet, you retrieve artifacts from specialized repositories dedicated to them, such as Docker® images contained in DockerHub, and Helm™ Charts that come from a dedicated Helm Chart repository. However, in an air-gapped environment, you need local repositories to store Helm charts, Docker images, and other artifacts. Tools such as JFrog™, Harbor™, and Nexus™ handle multiple types of artifacts in one local repository.

3.1.6.1.1 Related Topic:

Pre-provisioned Infrastructure (see page 1070)

3.1.6.2 Next Topic:

Cluster Types and Concepts (see page 79)

3.2 Licenses

This section contains overviews of the feature sets for all the available DKP licenses and how to acquire and use your license.

- DKP Essential (see page 91) supports your single-cluster environment, or a single Management cluster.
- DKP Enterprise (see page 89) supports a multi-cluster environment, with a Management Cluster and one or more Attached or Managed Clusters.
- DKP Government Essential (see page 96) consists of DKP Essential with Confirmed State Side Support (CSS)
- DKP Government Advanced (see page 93) consists of DKP Enterprise with Confirmed State Side Support (CSS)

All DKP Enterprise and DKP Government Advanced features are marked with the following badges:

Enterprise Gov Advanced

The type of license you have determines what DKP features are available to you. Some features that are compatible with all versions of DKP can be activated by purchasing additional Add-on licenses. The following table shows the differences between the different DKP licenses:

Feature	DKP Essential	DKP Enterprise	DKP Gov Essential	DKP Gov Advanced
Applications				
Workspace Platform Applications	✅	✅	✅	✅
Project Platform Applications (see page 719)		✅		✅
Catalog Applications (Kafka and Zookeeper)		✅		✅
Custom Applications		✅		✅
DKP Insights[54]				
Cluster Management				
Single Cluster	✅	✅	✅	✅
Multiple Clusters		✅		✅
Attaching Cluster		✅		✅
Provisioning an additional cluster		✅		✅
Upgrades (see page 1705)	✅	✅	✅	✅
GitOps				

54 https://d2iq.atlassian.net/wiki/spaces/DINS

Feature	DKP Essential	DKP Enterprise	DKP Gov Essential	DKP Gov Advanced
Continuous Deployment (FluxCD)		✅		✅
UX				
DKP CLI (see page 1835)	✅	✅	✅	✅
DKP UI	✅	✅	✅	✅
Workspaces		✅		✅
Projects		✅		✅
Logging				
Workspace Level Logging	✅	✅	✅	✅
Multi-tenant Logging		✅		✅
Monitoring	✅	✅	✅	✅
Backup & Restore (see page 904)	✅	✅	✅	✅
GPU	✅	✅	✅	✅
Security				

Feature	DKP Essential	DKP Enterprise	DKP Gov Essential	DKP Gov Advanced
Authentication using Dex	✅	✅	✅	✅
Policy control using Gatekeeper	✅	✅	✅	✅
Cluster Provisioning				
DKP on AWS (see page 294)	✅	✅	✅	✅
DKP on Azure (see page 541)	✅	✅	✅	✅
DKP on GCP (see page 571)	✅	✅	✅	✅
DKP on vSphere (see page 441)	✅	✅	✅	✅
Pre-provisioned (see page 148)	✅	✅	✅	✅
On-Prem (see page 148)	✅	✅	✅	✅
EKS Provisioning (see page 412)		✅		✅
AKS Provisioning (see page 556)		✅		✅
FIPS Compliance	✅	✅	✅	✅
Konvoy Image Builder (see page 1626)	✅	✅	✅	✅
Air-Gapped Deployments	✅	✅	✅	✅
Confirmed Stateside Support (CSS)			✅	✅

3.2.1 Buy a License

DKP Licenses are sold in units of cores. To learn more about both, and to obtain a valid license:

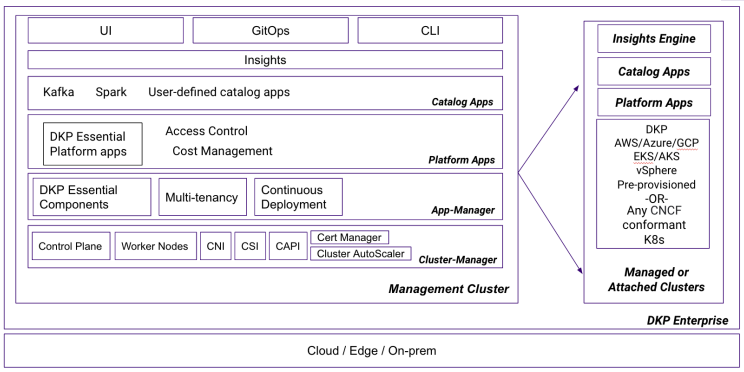
- Contact your sales representative at sales@d2iq.com[55] to buy one or more licenses.
- Buy a license via the AWS Marketplace[56], then the information (ARN) is accessible in the AWS License Manager console.

Next, download the binary files (see page 76).

3.2.2 DKP Enterprise

Enterprise Gov Advanced

Features and capabilities that are specific to DKP Enterprise are marked with this badge at the top of each page. These items are NOT available to DKP Essential customers.



DKP Enterprise is a multi-cluster lifecycle management Kubernetes solution centered around a management cluster that manages many attached or managed Kubernetes clusters via a centralized management dashboard. The management dashboard gives you a single point of observability and control throughout all of your attached or managed clusters. The DKP Enterprise license gives the customer access to the entire Konvoy cluster environment, the DKP UI dashboard that deploys platform and catalog applications, and multi-cluster management, and comprehensive compatibility with our full range of infrastructure deployment options.

3.2.2.1 Compatible Infrastructure

DKP Enterprise operates across D2iQ's entire range of cloud, on-premises, edge, and air-gapped infrastructures and has support for various OSs, including immutable OSs. See Supported Operating Systems (see page 67) for a full list of compatible infrastructure.

For the basics on standing up a DKP Enterprise cluster in one of the listed environments of your choice, see Advanced Configuration (see page 1598).

55 mailto:sales@d2iq.com
56 https://aws.amazon.com/marketplace/

3.2.2.2 Platform Applications

Applications can be deployed in any DKP managed cluster, giving you complete flexibility to operate across cloud, on-premises, edge, and air-gapped scenarios. Customers can also use the UI with Kommander to customize which platform applications to deploy to the cluster in a given workspace.

3.2.2.3 Catalog Applications

Quickly and easily deploy applications and complex data services from a centralized service catalog to specific or multiple clusters, with governance. Fast data pipelines can be provisioned automatically from the following catalog of DKP Applications:

- **Kafka:** Primarily used to build real-time streaming data pipelines and applications that adapt to the data streams. It combines messaging, storage, and stream processing to allow storage and analysis of both historical and real-time data.
- **Zookeeper:** A centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

For instructions on how to deploy catalog applications, see Workspace Catalog Applications (see page 691)

3.2.2.4 Cluster Manager

Konvoy is the Kubernetes installer component of DKP Enterprise that uses industry standard tools to produce a certified Kubernetes cluster. These industry standard tools create a cluster management system that includes:

- **Control Plane:** Manages the worker nodes and pods in the cluster.
- **Worker Nodes:** Used to run containerized applications and handle networking to ensure that traffic between applications across the cluster and from outside of the cluster can be properly facilitated.
- **Container Networking Interface (CNI):** Calico's open source networking and network security solution for containers, virtual machines, and native host-based workloads.
- **Container Storage Interface (CSI):** A common abstraction to container orchestrators for interacting with storage subsystems of various types.
- **Kubernetes Cluster API (CAPI):** Cluster API uses Kubernetes-style APIs and patterns to automate cluster lifecycle management for platform operators. For more on how CAPI is integrated into DKP Enterprise, see Understanding CAPI Concepts and Terms (see page 82)
- **Cert Manager:** A Kubernetes addon to automate the management and issuance of TLS certificates from various issuing sources.
- **Cluster Autoscaler:** A component that automatically adjusts the size of a Kubernetes cluster so that all pods have a place to run and there are no unneeded nodes.

3.2.2.5 Built-in GitOps

DKP Enterprise comes bundled with GitOps, an operating model for Kubernetes and other cloud native technologies, providing a set of best practices that unify Git deployment, management and monitoring for containerized clusters and applications. GitOps works by using Git as a single source of truth for declarative infrastructure and applications. With GitOps, the use of software agents can alert on any divergence between

Git with what is running in a cluster, and if there's a difference, Kubernetes reconcilers automatically update or rollback the cluster depending on the case.

3.2.2.6 DKP Enterprise Multi-cluster UI

Bundled with DKP Enterprise is a multi-cluster management view in the UI with Kommander that can be used in lieu of the bundled CLI. From the UI you can:

- **Connect to an infrastructure provider:** DKP Enterprise supports on-premises and cloud infrastructure providers such as AWS and Azure for your Konvoy clusters. To automate their provisioning, DKP requires authentication keys to your preferred infrastructure provider entered on the Add Infrastructure Provider form.
- **Setup identity providers:** DKP Enterprise supports GitHub, LDAP, SAML, and standard OIDC identity providers such as Google. These identity management providers support the login and authentication process for your Kubernetes cluster. See Identity Providers (see page 609) for more information.
- **Configure access control:** Role-based authorization control (RBAC) is central to DKP Enterprise and controls access to resources on all connected clusters. The resources are similar to Kubernetes RBAC. You add an identity provider group, and in that group add cluster roles and cluster role bindings for those roles.
- **Deploy applications:** The DKP Enterprise UI allows you to customize your workspace application deployments via the Applications page within the UI.
- **Create a project:** Create projects within a workspace and deploy project-scoped applications. Projects enable teams to deploy configurations and services to their clusters consistently. After configuring roles, ConfigMaps, secrets, and applications for a project, DKP distributes this configuration to each project namespace. For more information concerning projects, see Projects (see page 716).
- **Add a license:** To add a license via the UI with Kommander, see Add a License (see page 98).
- **Kubernetes cost monitoring and management**: The kubecost platform application provides real-time cost visibility and insights for external cloud services such as AWS, helping you continuously reduce your cloud costs.

For more information concerning the global and workspace-level UI with Kommander, see Workspaces (see page 678).

3.2.3 DKP Essential

DKP Essential is a self-managed single cluster Kubernetes solution that gives you a feature-rich, easy-to-deploy, and easy-to-manage entry-level cloud container platform. The DKP Essential license gives the user access to the entire Konvoy cluster environment, and to the Kommander platform application manager.

Features and capabilities that are specific to DKP Enterprise are marked with this badge at the top of each page. These items are NOT available to DKP Essential users.

3.2.3.1 Compatible Infrastructure

DKP Essential operates across a range of cloud, on-premise, edge, and air-gapped infrastructures and has support for various OSs, including immutable OSs. See Supported Operating Systems (see page 67) for a full list of compatible infrastructure.

For the basics on standing up a DKP Essential cluster in one of the listed environments of your choice, see Advanced Configuration (see page 1598).

3.2.3.2 Platform Applications

When creating a cluster, the application manager deploys certain platform applications on the newly created cluster. DKP Essential users can use the Kommander UI to customize which platform applications to deploy to the cluster in a given workspace. For a list of available platform applications that are included with DKP, see Workspace Platform Applications (see page 124).

> **NOTE:** The platform application `kubecost` is not included with DKP Essential, but is included with DKP Enterprise.

3.2.3.3 Cluster Manager

Konvoy is the Kubernetes installer component of DKP Essential that uses industry standard tools to produce a certified Kubernetes cluster. These industry standard tools create a cluster management system that includes:

- **Control Plane:** Manages the worker nodes and pods in the cluster.
- **Worker Nodes:** Used to run containerized applications and handle networking to ensure that traffic between applications across the cluster and from outside of the cluster can be properly facilitated.
- **Container Networking Interface (CNI):** Calico's open source networking and network security solution for containers, virtual machines, and native host-based workloads.
- **Container Storage Interface (CSI):** A common abstraction to container orchestrators for interacting with storage subsystems of various types.
- **Kubernetes Cluster API (CAPI):** Cluster API uses Kubernetes-style APIs and patterns to automate cluster lifecycle management for platform operators. For more on how CAPI is integrated into DKP Essential, see Understanding CAPI Concepts and Terms (see page 82)
- **Cert Manager:** A Kubernetes addon to automate the management and issuance of TLS certificates from various issuing sources.
- **Cluster Autoscaler:** A component that automatically adjusts the size of a Kubernetes cluster so that all pods have a place to run and there are no unneeded nodes.

3.2.3.4 Built-in GitOps

DKP Essential comes bundled with GitOps, an operating model for Kubernetes and other cloud native technologies, providing a set of best practices that unify Git deployment, management and monitoring for containerized clusters and applications. GitOps works by using Git as a single source of truth for declarative infrastructure and applications. With GitOps, the use of software agents can alert on any divergence between Git with what is running in a cluster, and if there's a difference, Kubernetes reconcilers automatically update or rollback the cluster depending on the case.

3.2.3.5 DKP Essential Single Cluster UI

Bundled with DKP Essential is a single cluster management view of DKP UI (see page 1556) that can be used in lieu of the bundled CLI. From the UI you can:

- **Setup identity providers:** DKP Essential supports GitHub, LDAP, SAML, and standard OIDC identity providers such as Google. These identity management providers support the login and authentication process for your Kubernetes cluster. See Identity Providers (see page 609) for more information.
- **Deploy applications:** The DKP Essential UI with Kommander allows you to customize your workspace application deployments via the Applications page within the UI.
- **Add a license:** To add a license via the UI with Kommander, see Add a license to Kommander (see page 98)

For more information concerning the global and workspace-level UI, see Workspaces (see page 678)

3.2.4 DKP Government Advanced

Features and capabilities that are specific to DKP Government Advanced are marked with this badge at the top of each page. These items are NOT available to DKP Government Essential users. In general, every feature available in DKP Enterprise (see page 89)is also available in DKP Government Advanced.

Enterprise Gov Advanced

DKP Government Advanced (also referred to as DKP Gov Advanced) is a multi-cluster Kubernetes management platform that is optimized for deployment in the government sector. With top-to-bottom declarative programming, DKP Government Advanced provides the most advanced and feature-rich Kubernetes management capability across any infrastructure, whether on-premise, in the cloud, in air-gapped environments, or at the edge.

Ease of use is the hallmark of D2iQ Kubernetes solutions. DKP Government Advanced enables you to be up and running in minutes and hours rather than weeks and months, with complete stability, reliability, military-grade security, and rapid time-to-value. Complexity is reduced through packaging, automation, integration, and elegant design.

Built on pure open-source Kubernetes, DKP Government Advanced enables you to avoid vendor lock-in, easily upgrade services, and take full advantage of open-source community innovation. DKP Government Advanced provides platform services for networking, storage, logging, monitoring, and more, all of which have been carefully selected from the Cloud Native Computing Foundation (CNCF) landscape and rigorously tested to work together.

DKP Government Advanced is a multi-cluster lifecycle management Kubernetes solution centered around a management cluster that manages multiple attached or managed Kubernetes clusters via a centralized

management dashboard. The management dashboard gives you a single point of observability and control throughout all of your attached or managed clusters. The DKP Government Advanced license gives the user access to the entire Konvoy cluster environment, the DKP UI dashboard that deploys platform and catalog applications, and multi-cluster management, and comprehensive compatibility with our full range of infrastructure deployment options.

3.2.4.1 Compatible Infrastructure

DKP Government Advanced operates across D2iQ's entire range of cloud, on-premises, edge, and air-gapped infrastructures and has support for various OSs, including immutable OSs. See Supported Operating Systems (see page 67) for a full list of compatible infrastructure.

For the basics on standing up a DKP Government Advanced cluster in one of the listed environments of your choice, see Additional Infrastructure Customized Configurations (see page 1050).

3.2.4.2 Platform Applications

Applications can be deployed in any DKP managed cluster, giving you complete flexibility to operate across cloud, on-premises, edge, and air-gapped scenarios. Customers can also use the UI with Kommander to customize which platform applications to deploy to the cluster in a given workspace.

3.2.4.3 Catalog Applications

Quickly and easily deploy applications and complex data services from a centralized service catalog to specific or multiple clusters, with governance. Fast data pipelines can be provisioned automatically from the following catalog of DKP Applications:

- **Kafka:** Primarily used to build real-time streaming data pipelines and applications that adapt to the data streams. It combines messaging, storage, and stream processing to allow storage and analysis of both historical and real-time data.
- **Zookeeper:** A centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

For instructions on how to deploy catalog applications, see Workspace Catalog Applications (see page 691)

3.2.4.4 Cluster Manager

Konvoy is the Kubernetes installer component of DKP Government Advanced that uses industry standard tools to produce a certified Kubernetes cluster. These industry standard tools create a cluster management system that includes:

- **Control Plane:** Manages the worker nodes and pods in the cluster.
- **Worker Nodes:** Used to run containerized applications and handle networking to ensure that traffic between applications across the cluster and from outside of the cluster can be properly facilitated.
- **Container Networking Interface (CNI):** Calico's open source networking and network security solution for containers, virtual machines, and native host-based workloads.
- **Container Storage Interface (CSI):** A common abstraction to container orchestrators for interacting with storage subsystems of various types.

- **Kubernetes Cluster API (CAPI):** Cluster API uses Kubernetes-style APIs and patterns to automate cluster lifecycle management for platform operators. For more on how CAPI is integrated into DKP Government Advanced, see Understanding CAPI Concepts and Terms (see page 82)
- **Cert Manager:** A Kubernetes addon to automate the management and issuance of TLS certificates from various issuing sources.
- **Cluster Autoscaler:** A component that automatically adjusts the size of a Kubernetes cluster so that all pods have a place to run and there are no unneeded nodes.

3.2.4.5 Built-in GitOps

DKP Government Advanced comes bundled with GitOps, an operating model for Kubernetes and other cloud native technologies, providing a set of best practices that unify Git deployment, management and monitoring for containerized clusters and applications. GitOps works by using Git as a single source of truth for declarative infrastructure and applications. With GitOps, the use of software agents can alert on any divergence between Git with what is running in a cluster, and if there's a difference, Kubernetes reconcilers automatically update or rollback the cluster depending on the case.

3.2.4.6 DKP Government Advanced Multi-cluster UI

Bundled with DKP Government Advanced is a multi-cluster management view in the UI with Kommander that can be used in lieu of the bundled CLI. From the UI you can:

- **Connect to an infrastructure provider:** DKP Government Advanced supports on-premises and cloud infrastructure providers such as AWS and Azure for your Konvoy clusters. To automate their provisioning, DKP requires authentication keys to your preferred infrastructure provider entered on the Add Infrastructure Provider form.
- **Setup identity providers:** DKP Government Advanced supports GitHub, LDAP, SAML, and standard OIDC identity providers such as Google. These identity management providers support the login and authentication process for your Kubernetes cluster. See Identity Providers (see page 609) for more information.
- **Configure access control:** Role-based authorization control (RBAC) is central to DKP Government Advanced and controls access to resources on all connected clusters. The resources are similar to Kubernetes RBAC. You add an identity provider group, and in that group add cluster roles and cluster role bindings for those roles.
- **Deploy applications:** The DKP Government Advanced UI allows you to customize your workspace application deployments via the Applications page within the UI.
- **Create a project:** Create projects within a workspace and deploy project-scoped applications. Projects enable teams to deploy configurations and services to their clusters consistently. After configuring roles, ConfigMaps, secrets, and applications for a project, DKP distributes this configuration to each project namespace. For more information concerning projects, see Projects (see page 716).
- **Add a license:** To add a license via the UI with Kommander, see Add a License (see page 98)
- **Kubernetes cost monitoring and management**: The kubecost platform application provides real-time cost visibility and insights for external cloud services such as AWS, helping you continuously reduce your cloud costs.

For more information concerning the global and workspace-level UI with Kommander, see Workspaces (see page 678)

3.2.4.7 Confirmed Stateside Support (CSS)

Because many governmental applications require CSS, DKP Government Advanced includes access to D2iQ's CSS team of U.S. citizens working at U.S. support centers.

3.2.4.8 Military-Grade Security

DKP Government Advanced can be configured to meet defined security standards. Each component is tested and certified before release. DKP also provides instant platform engineering that reduces the burden of security on DevOps and DevSecOps teams. Being based on pure upstream open-source Kubernetes enables easy and trouble-free upgrades, patches, and bug fixes.

3.2.5 DKP Government Essential

DKP Government (Gov) Essential is a single-cluster Kubernetes management platform that is optimized for deployment in the government sector. Based on the D2iQ Kubernetes Platform (DKP), DKP Government Essential can easily and seamlessly expand to a multi-cluster Kubernetes management platform as your needs grow.

With top-to-bottom declarative programming, DKP Government Essential provides the most advanced and feature-rich single-cluster Kubernetes management capability across any infrastructure, whether on-premise, in the cloud, in air-gapped environments, or at the edge.

Ease of use is the hallmark of D2iQ Kubernetes solutions. DKP Government Essential enables you to be up and running in minutes and hours rather than weeks and months, with complete stability, reliability, military-grade security, and rapid time-to-value. Complexity is reduced through packaging, automation, integration, and elegant design.

Built on pure open-source Kubernetes, DKP Government Essential enables you to avoid vendor lock-in, easily upgrade services, and take full advantage of open-source community innovation. DKP Government Essential provides platform services for networking, storage, logging, monitoring, and more, all of which have been carefully selected from the Cloud Native Computing Foundation (CNCF) landscape and rigorously tested to work together.

DKP Government Essential is a self-managed single cluster Kubernetes solution that gives you a feature-rich, easy-to-deploy, and easy-to-manage entry-level cloud container platform. The DKP Government Essential license gives the user access to the entire Konvoy cluster environment, and to the Kommander platform application manager. In general, every feature available in DKP Essential (see page 91)is also available in DKP Government Essential

3.2.5.1 Compatible Infrastructure

DKP Government Essential operates across a range of cloud, on-premise, edge, and air-gapped infrastructures and has support for various OSs, including immutable OSs. See Supported Operating Systems (see page 67) for a full list of compatible infrastructure.

For the basics on standing up a DKP Government Essential cluster in one of the listed environments of your choice, see Additional Infrastructure Configurations (see page 1050).

3.2.5.2 Platform Applications

When creating a cluster, the application manager deploys certain platform applications on the newly created cluster. DKP Government Essential users can use the Kommander UI to customize which platform applications to deploy to the cluster in a given workspace. For a list of available platform applications that are included with DKP, see Workspace Platform Applications (see page 124).

> **NOTE:** The platform application `kubecost` is not included with DKP Government Essential, but is included with DKP Enterprise (see page 91) and DKP Government Advanced (see page 93).

3.2.5.3 Cluster Manager

Konvoy is the Kubernetes installer component of DKP Government Essential that uses industry standard tools to produce a certified Kubernetes cluster. These industry standard tools create a cluster management system that includes:

- **Control Plane:** Manages the worker nodes and pods in the cluster.
- **Worker Nodes:** Used to run containerized applications and handle networking to ensure that traffic between applications across the cluster and from outside of the cluster can be properly facilitated.
- **Container Networking Interface (CNI):** Calico's open source networking and network security solution for containers, virtual machines, and native host-based workloads.
- **Container Storage Interface (CSI):** A common abstraction to container orchestrators for interacting with storage subsystems of various types.
- **Kubernetes Cluster API (CAPI):** Cluster API uses Kubernetes-style APIs and patterns to automate cluster lifecycle management for platform operators. For more on how CAPI is integrated into DKP Government Essential, see Understanding CAPI Concepts and Terms (see page 82)
- **Cert Manager:** A Kubernetes addon to automate the management and issuance of TLS certificates from various issuing sources.
- **Cluster Autoscaler:** A component that automatically adjusts the size of a Kubernetes cluster so that all pods have a place to run and there are no unneeded nodes.

3.2.5.4 Built-in GitOps

DKP Government Essential comes bundled with GitOps, an operating model for Kubernetes and other cloud native technologies, providing a set of best practices that unify Git deployment, management and monitoring for containerized clusters and applications. GitOps works by using Git as a single source of truth for declarative infrastructure and applications. With GitOps, the use of software agents can alert on any divergence between Git with what is running in a cluster, and if there's a difference, Kubernetes reconcilers automatically update or rollback the cluster depending on the case.

3.2.5.5 DKP Government Essential Single Cluster UI

Bundled with DKP Government Essential is a single cluster management view of DKP UI (see page 1556) that can be used in lieu of the bundled CLI. From the UI you can:

- **Setup identity providers:** DKP Government Essential supports GitHub, LDAP, SAML, and standard OIDC identity providers such as Google. These identity management providers support the login and authentication process for your Kubernetes cluster. See Identity Providers (see page 609) for more information.
- **Deploy applications:** The DKP Government Essential UI with Kommander allows you to customize your workspace application deployments via the Applications page within the UI.
- **Add a license:** To add a license via the UI with Kommander, see Add a license to Kommander (see page 98)

For more information concerning the global and workspace-level UI, see Workspaces (see page 678)

3.2.5.6 **Confirmed Stateside Support (CSS)**

Because many governmental applications require CSS, DKP Government Essential includes access to D2iQ's CSS team of U.S. citizens working at U.S. support centers.

3.2.5.7 **Military-Grade Security**

DKP Government Essential can be configured to meet defined security standards. Each component is tested and certified before release. DKP also provides instant platform engineering that reduces the burden of security on DevOps and DevSecOps teams. Being based on pure upstream open-source Kubernetes enables easy and trouble-free upgrades, patches, and bug fixes.

3.2.6 Add a DKP license

3.2.6.1 Prerequisites

For licenses that you buy from the AWS Marketplace[57], an AWS administrator must attach the AWS managed policy[58] `AWSLicenseManagerConsumptionPolicy` to the `control-plane.cluster-api-provider-aws.sigs.k8s.io` role created when configuring AWS IAM policies[59]. If an Administrator does not attach this policy attached to the role, DKP cannot verify the license information provided in the procedure steps that follow.

The machine onto which you install DKP must meet these requirements:

57 https://aws.amazon.com/marketplace/
58 https://docs.aws.amazon.com/license-manager/latest/userguide/security-iam-awsmanpol.html#security-iam-AWSLicenseManagerConsumptionPolicy
59 https://d2iq.atlassian.net/wiki/spaces/ES/pages/9897837/Configure+AWS+IAM+Policies

- Docker installed - DKP uses Kubernetes-in-Docker (Kind) to create a bootstrap cluster for creating Management clusters, and thus Docker installation is required. Docker is not used to run your Management or workload clusters.
- Active Internet connectivity.
- Existing AWS account.
- AWS Administrator logged into the AWS account used to buy DKP.

3.2.6.2 Download the Container Image and Extract the Binaries

1. When you complete the sales transaction in the AWS Marketplace, select a DKP version from the **Choose a fulfillment option** dropdown. We recommend that you select the latest version.
2. Select **Continue to Launch** to obtain the download instructions, available when you select **Usage Instructions**.
3. Complete the steps in each of the procedures under **Usage instructions**. The links in these steps take you to DKP documentation pages that provide the steps you need to set up AWS to work with DKP, and the download and installation of DKP itself.
4. Expand the **Create a license token and IAM role** link in the **Container images** section of the window. This gives you access to the Create token button to generate the license token and IAM role and exposes an inset code window with the token and role credentials.
5. Use the generated command to log in to AWS using the token and role you just generated. Note that you can omit this step if you have already configured the AWS CLI with valid credentials.
   Tip: Using the Copy button at the upper right of this inset code window helps ensure you copy the entire command.
6. Copy the commands at the bottom of the page, and run them sequentially to download the container images. It will look similar to this:

   ```
   aws ecr get-login-password \
     --region <insert-region> | docker login \
     --username <username> \
     --password-stdin <ecr-address>
   ```

   ```
   CONTAINER_IMAGES="<ecr-address>/mesosphere/d2iq-dkp-essential-premium-
   support:v2.5.0"
   ```

   ```
   for i in $(echo $CONTAINER_IMAGES | sed "s/,/ /g"); do docker pull $i; done
   ```

7. Access a terminal window on the Linux machine where you plan to run the DKP CLI cluster, and execute the command above, using your environment's specific values. This downloads a container that contains the DKP binary.

8. After downloading the container, run the following command to copy the binaries onto your Linux machine.

   ```
   docker run -it --rm -u $(id -u):$(id -g) -v $(pwd):/dkp $CONTAINER_IMAGES
   ```

   You will then see the following output:

   ```
   dkp binary is placed in the local directory, to run:
   ./dkp --help
   ```

   You will now see the `dkp` binary in your working directory. Follow the DKP installation (see page 127) instructions using these binaries, and then add your license (see page 98) to DKP.

3.2.6.3 Obtain the Amazon Resource Number (ARN) from the AWS Marketplace UI

You must use the AWS Marketplace user interface to navigate to the license table and configure its settings to display the ARN for your purchased license.

1. From the **Launch this software** page, navigate back to the **Product Detail** page.
2. Select the **View Subscription** button at the upper right (in the blue information box labeled "***You have access to this product***").
3. Select **Manage subscriptions** from the left-hand navigation pane, and then click the **Manage** button on the specific license. The details page for that license displays.
4. Select the **License** number hyperlink to display the license page, then select **Granted licenses** from the left-hand navigation pane and find your license in the list.
5. Use the **Settings** icon in the upper right corner to display the columns displayed, and then enable the **License arn** value.
6. Select the **Confirm** button to return to the license list with the License ARN displayed in the last column. You can now copy and paste this ARN as needed.

If you have difficulty obtaining your ARN, contact D2iQ Support for assistance.

3.2.6.4 Enter License Information in the DKP UI

For licenses bought directly from D2iQ, you can obtain the license token through the Support Portal[60]. Insert this token in the last step below.

For licenses purchased through the AWS marketplace, you can find the license in the "Granted Licenses" table of AWS License Manager inside the AWS console. If necessary, modify the table view preferences to show the "License ARN." Copy this value for use in the last step below.

60 https://support.d2iq.com/hc/en-us

You must be an administrator to add licenses to DKP.

From the the UI with Kommander, complete the following steps:

1. Select **Global** in the workspace header drop-down.
2. In the sidebar menu, select **Administration** > **Licensing**.
3. Select + Activate License to enter the Activate License form.
4. On the Activate License form page, select **D2iQ** or **AWS Marketplace** depending on where you acquired your license.
5. Paste your license token, or the AWS ARN, in the text box and select **Save**.

If there is an error submitting a license acquired directly from D2iQ, you can activate the license directly through `kubectl`.

3.2.6.5 Enter a DKP License using kubectl

You can activate a license acquired from D2iQ directly using the `kubectl` utility.

1. Create a secret, replacing the value `MY_LICENSE` in the command that follows with your actual, D2iQ-provided DKP license token:

```
kubectl create secret generic my-license-secret --from-literal=jwt=MY_LICENSE
-n kommander
kubectl label secret my-license-secret kommanderType=license -n kommander
```

2. Create a license object with this command:

```
cat <<EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: License
metadata:
  name: my-license
  namespace: kommander
spec:
  licenseRef:
    name: my-license-secret
EOF
```

3. Return to the license page in the DKP UI to see your valid license display.

3.2.6.6 Activate a DKP Insights License Key

Instructions for activating an Insights License Key in the DKP UI

3.2.6.6.1 Prerequisites

Prior to activating an Insights license key, ensure that you have a valid DKP Essential or DKP Enterprise license.

3.2.6.6.2 Obtain a License Key

DKP Insights license keys to activate the product are provided directly by D2iQ. Contact your account rep to receive a license key.

After you receive a license key, follow the instructions in the next section to activate DKP Insights in your environment.

3.2.6.6.3 Enter License Information

An administrator must activate licenses in DKP.

From the DKP UI, complete the following steps:

1. Select **Global** in the workspace header drop-down.
2. In the sidebar menu, select **Administration** > **Licensing**.
3. Select **+ Activate Add-on License**.
4. In the Add-on License form page, input the Insight license key that was provided to you by DKP.
5. Select **Save**.

Once you activate an Insights license key, the prompt to activate an Add-on License is removed.

3.2.6.6.4 Remove an Insights License

If your Insights license information has changed, you may need to remove an existing license from DKP to activate a new one. Only DKP administrators can remove licenses.

In the DKP UI, perform the following:

1. Select **Global** in the workspace header drop-down.
2. In the sidebar menu, select **Administration** > **Licensing**.
3. Select **Remove License** in the DKP Insights card, then follow the prompts.

3.2.7 Remove a DKP license

If your license information has changed, you may need to remove an existing license from DKP to add a new one. Only DKP administrators have the ability to remove licenses.

> Original license information can still be obtained from D2iQ or the AWS License Manager console even after removing from DKP.

3.2.7.1 Remove a License via the UI

In the DKP UI, do the following:

1. Select **Global** in the workspace header drop-down.
2. In the sidebar menu, select **Administration** > **Licensing**.
3. Your existing licenses will be listed. Click **Remove License** on the license you would like to remove, and follow the prompts.

3.2.7.2 Manually Remove a License using kubectl

To remove a license from DKP using `kubectl`, you have to delete the `Secret` and `License` objects. In this example, the secret is named “my-license-secret”.

1. Validate that the secret exists in the `kommander` namespace:

   ```
   kubectl describe secret -n kommander my-license-secret
   ```

   Expected output:

   ```
   Name:         my-license-secret
   Namespace:    kommander
   Labels:       kommanderType=license
   Annotations:  <none>

   Type:         Opaque
   ```

```
Data
====
jwt:  455 bytes
```

2. Delete the secret from the `kommander` namespace:

```
kubectl delete secret -n kommander my-license-secret
```

Expected output:

```
secret "my-license-secret" deleted
```

3. We do the same with the `License` object. Validate that it exists in the `kommander` namespace:

```
kubectl describe license -n kommander my-license
```

Expected output:

```
Name:         my-license
Namespace:    kommander
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"kommander.mesosphere.io/v1beta1","kind":"License
","metadata":{"annotations":{},"name":"my-license", "namespace":"kommand...
API Version:  kommander.mesosphere.io/v1beta1
Kind:         License
Metadata:
  Creation Timestamp:  2020-03-25T14:57:31Z
  Generate Name:       license-
  Generation:          1
  Resource Version:    17895
  Self Link:           /apis/kommander.mesosphere.io/v1beta1/namespaces/
kommander/licenses/my-license
  UID:                 35ee9254-4094-40eb-a2d8-4687c5d212d9
Spec:
  License Ref:
    Name:  my-license-secret
Status:
  Cluster Capacity:  500
  Customer Id:       mesosphere-developer
  End Date:          2020-10-02T14:00:09Z
  License Id:        mesosphere-developer
  Start Date:        2019-10-02T14:00:09Z
  Valid:             true
  Version:           1.0
Events:
```

```
  Type    Reason                Age                  From              Message
  ----    ------                ----                 ----              -------
  Normal  LicenseUpdateSuccess  7m7s (x2 over 7m7s)  LicenseSignature  License
updated successfully
```

4. Delete the license from the `kommander` namespace:

```
kubectl delete license -n kommander my-license
```

Expected output:

```
license.kommander.mesosphere.io "my-license" deleted
```

You have now successfully removed a license.

3.2.8 MSP Program

The D2iQ Managed Service Provider (MSP) Program is a partnership program targeted at companies that provide cloud infrastructure services to their customers. When becoming a member, you can benefit from flexible billing options and enhanced support. For more information, read the full press release[61].

3.2.8.1 Benefits

- Transparent and flexible billing options
- Enhanced support experience
- Support inquiries are routed to CKA-certified, senior support engineers

3.2.8.2 How does it work?

After you sign your MSP agreement, you will be part of the program.

To benefit from the flexible billing options, submit a monthly report in the form of a **telemetry bundle**[62] for any cluster that ran during the month.

3.2.8.3 How do I become an MSP partner?

Contact your D2iQ representative or use our contact form[63] to obtain more information on becoming a partner.

61 https://d2iq.com/press-release/managed-service-providers-kubernetes-management-program
62 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/223543301/Sending+Telemetry+Data+for+Analysis
63 https://d2iq.com/contact

3.3 Provide Context for Commands with a kubeconfig File

This page contains some basic recommendations regarding `kubeconfig` as it relates to target clusters and the `--kubeconfig=<CLUSTER_NAME>.conf` flag. Refer to the Kubernetes documentation[64] for more information.

For `kubectl` and `dkp` commands to run, it is often necessary to specify the environment or cluster in which you want to run them. This specially applies to commands that create, delete, or update a cluster's resources.

To specify the context, there are two options:

Export an environment variable	Specify the target cluster in the command
Export an environment variable from a cluster's `kubeconfig` file, which sets the environment for the commands you run after exporting it.	Specify an environment variable for one command at the time by running it with the `--kubeconfig=<CLUSTER_NAME>.conf` flag.
• Better suited for single-cluster environments	• Better suited for multi-cluster environments

3.3.1 Single-cluster Environment

In a single-cluster environment, you don't need to switch between clusters to execute commands and perform operations. However, it is still necessary to specify an environment for each terminal session, so the DKP CLI runs the operations on the DKP cluster, and does not accidentally run operations on, for example, the bootstrap cluster.

Use a kubeconfig file to set the environment variable for all your operations

3.3.1.1 Set the environment variable for all your operations

1. When you create a cluster, a `kubeconfig` file is generated automatically. Get this `kubeconfig` file and write it to the `${CLUSTER_NAME}.conf` variable:

64 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

After setting this configuration, whenever you use run a command with the variable `${CLUSTER_NAME}.conf`, it will automatically run on your cluster.

2. Set the context by exporting your `kubeconfig` from the source file:
:note: Execute this command for each terminal session.

```
export KUBECONFIG=${CLUSTER_NAME}.conf
```

For the current terminal session, you don't need to provide the `--kubeconfig` flag.

3.3.2 Multi-cluster Environment

Since having multiple clusters means switching between them to run operations, D2iQ recommends two approaches.

A. You can start several terminal sessions, one per cluster, and set the environment variable as shown in the single-cluster environment example (see page 106) above, one time per cluster.

B. You can use a single terminal session, and run the commands with a flag every time. The flag specifies the target cluster for the operation every time, so you can run the same command several times but with a different flag. Expand the following content for an explanation of option B.

B. Use a flag to reference the target cluster

3.3.2.1 Use a flag to reference the target cluster

The `--kubeconfig=<CLUSTER_NAME>.conf` flag defines the configuration file for the cluster that you are configuring and accessing.

When operating and using multiple clusters, this is the easiest way to ensure that you are working on the correct cluster. If you create additional clusters and do not store the name as an environment variable, you can type out the name of the cluster followed by `.conf` to access your cluster.

Ensure you run `dkp get kubeconfig` for each cluster you create to generate a `kubeconfig` file.

Example flag:

```
--kubeconfig=azurecluster1.conf
```

Example command with flag:

```
dkp install kommander --kubeconfig=azurecluster1.conf
```

Advanced: Use a kubeconfig file to configure access to multiple clusters using contexts

For advanced users only.

It is possible to set up a `kubeconfig` file to manage access to several clusters. For more information, refer to https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/.

You can also set your environment variable to multiple `kubeconfig` files by merging them. For more information, refer to https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/.

3.3.3 Next Topic:

Storage (see page 108)

3.4 Storage

3.4.1 An Introduction to Persistent Storage in Kubernetes

This document describes the model used in Kubernetes for managing persistent, cluster-scoped storage for workloads requiring access to persistent data.

A workload on Kubernetes typically requires two types of storage:

- Ephemeral Storage
- Persistent Volume

3.4.1.1 Ephemeral Storage

Ephemeral storage, by its name, is ephemeral in the sense that it is cleaned up when the workload is deleted or the container crashes. For example, the following are examples of ephemeral storage provided by Kubernetes:

EmptyDir volume.	Managed by kubelet under `/var/lib/kubelet`.
Container logs.	Typically under `/var/logs/containers`.

Container image layers.	Managed by container runtime (e.g., under `/var/lib/containerd`).
Container writable layers.	Managed by container runtime (e.g., under `/var/lib/containerd`).

Ephemeral storage is automatically managed by Kubernetes, and typically does not require explicit settings. You may need to express the capacity requests for ephemeral storage so that `kubelet` can use that information to make sure it does not run out of ephemeral storage space on each node.

3.4.1.2 Persistent Volume

Persistent Volumes are storage resources that can be used by the cluster. Persistent Volumes are volume plug-ins that have lifecycle capabilities that are independent of any Kubernetes Pod or Deployment. A Kubernetes persistent volume (PV) is an object that allows pods to access persistent storage on a storage device and defined via a Kubernetes StorageClass. Unlike regular volumes, which are transient in nature, PVs are persistent, supporting stateful application use cases.

You may have stateful workloads requiring persistent storage whose lifecycle is longer than that of Pods or containers. For instance, a database server needs to recover database files after it crashes. For those cases, the workloads need to use PersistentVolumes (PV).

Persistent Volumes are resources that represent storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. Unlike ephemeral storage, the lifecycle of a PersistentVolume is independent of that of the workload that uses it.

The Persistent Volume API objects capture the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system. In order to use a Persistent Volume (PV), your application needs to invoke a Persistent Volume Claim (PVC).

3.4.1.3 Persistent Volume Claim

A PersistentVolumeClaim is a request for storage. For a workload that requires persistent volumes, the workload should use PersistentVolumeClaim (PVC) to express its request on persistent storage. A PersistentVolumeClaim can request specific size and Access Modes[65] (for example, they can be mounted after read/write or many times read-only).

Any workload can specify a PersistentVolumeClaim. For example, a Pod may need a volume that is at least 4Gi large or a volume mounted under `/data` in the container's filesystem. If there is a PersistentVolume (PV) that satisfies the specified requirements in the PersistentVolumeClaim (PVC), it will be bound to the PVC before the Pod starts.

65 https://kubernetes.io/docs/concepts/storage/persistent-volumes/#access-modes

3.4.1.4 Related Information

- Storage for Applications (see page 1034) in the Kommander component
- Kubernetes Storage[66]
- Kubernetes persistent storage design document[67]
- Default Storage Providers in DKP (see page 110)

3.4.1.5 Next Step:

Resource Requirements (see page 119)

3.4.2 Default Storage Providers in DKP

Default storage providers in DKP

When deploying DKP using a supported cloud provisioner (AWS, Azure, GCP), DKP automatically configures native storage drivers for the target platform. In addition, DKP deploys a default StorageClass[68] for https://kubernetes.io/docs/concepts/storage/dynamic-provisioning/ creation. The table below lists the driver and default StorageClass for each supported cloud provisioner.

Cloud Provisioner	Version	Driver	Default Storage Class
AWS	`1.23` The AWS CSI driver is upgraded to a new minor version in DKP 2.7, which	aws-ebs-csi-driver[70]	`ebs-sc`

66 https://kubernetes.io/docs/concepts/storage/
67 https://github.com/kubernetes/design-proposals-archive/tree/main/storage
68 https://kubernetes.io/docs/concepts/storage/storage-classes/
70 https://github.com/kubernetes-sigs/aws-ebs-csi-driver

Cloud Provisioner	Version	Driver	Default Storage Class
	has an upgraded version[69] of the package.		
Azure	`1.29.0`	azuredisk-csi-driver[71]	`azuredisk-sc`
Pre-provisioned	`2.6.0`	local-static-provisioner[72]	`localvolumeprovisioner`
vSphere	`2.7.2`	vsphere-csi-driver[73]	`vsphere-raw-block-sc`
Cloud Director (VCD)	`1.4.0-d2iq.0`	cloud-director-named-csi-driver[74]	`vcd-disk-sc`
GCP	`1.10.3`	gcp-compute-persistent-disk-csi-driver[75]	`csi-gce-pd`

⚠ DKP uses the local static provisioner as the default storage provider for Pre-provisioned clusters. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[76] compatible storage that is suitable for production.

69 https://github.com/kubernetes-sigs/aws-ebs-csi-driver/blob/master/CHANGELOG.md#urgent-upgrade-notes
71 https://github.com/kubernetes-sigs/azuredisk-csi-driver
72 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner
73 https://github.com/kubernetes-sigs/vsphere-csi-driver
74 https://github.com/vmware/cloud-director-named-disk-csi-driver
75 https://github.com/kubernetes-sigs/gcp-compute-persistent-disk-csi-driver/releases
76 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

You can choose from any of the storage options[77] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[78].

When a default StorageClass is specified, you can create **PersistentVolumeClaims** (PVCs) without needing to specify the storage class. For instance, to request a volume using the default provisioner, create a PVC with the following:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 4Gi
```

To start the provisioning of a volume, launch a pod which references the PVC:

```
...
    volumeMounts:
    - mountPath: /data
      name: persistent-storage
...
  volumes:
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: my-pv-claim
```

To specify a StorageClass that references a storage policy when making a PVC[79], you can name it in `storageClassName`. If left blank, the default StorageClass is used.

77 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
78 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/
79 https://kubernetes.io/docs/concepts/storage/persistent-volumes/#class-1

3.4.2.1 Change or Manage Multiple StorageClasses

The default `StorageClass` provisioned with DKP is acceptable for install, but may not suitable for production. If your workload has different requirements, you can create additional `StorageClass` types with specific configurations. You can change the default `StorageClass` using these steps from the Kubernetes site:

- Changing the default storage class[80]

Ceph can also be used as CSI storage. Refer to that section in the documentation for information on how to use Rook Ceph (see page 1034).

3.4.2.2 Driver Information

All default drivers implement the Container Storage Interface[81] (CSI). The CSI provides a common abstraction to container orchestrators for interacting with storage subsystems of various types. Each driver has specific configuration parameters which effect PV provisioning. This section details the default configuration for drivers used with DKP. This section also has links to driver documentation, if further customization is required.

NOTE: `StorageClass` parameters cannot be changed after creation. To use a different volume configuration, you must create a new `StorageClass`.

3.4.2.2.1 Amazon Elastic Block Store (EBS) CSI Driver

DKP EBS default `StorageClass`:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true" # This tells kubernetes to
make this the default storage class
  name: ebs-sc
provisioner: ebs.csi.aws.com
reclaimPolicy: Delete  # volumes are automatically reclaimed when no longer in use
and PVCs are deleted
```

80 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/
81 https://github.com/container-storage-interface/spec/blob/master/spec.md

```
volumeBindingMode: WaitForFirstConsumer #  Physical volumes will not be created until
a pod is created that uses the PVC, required to use CSI's Topology feature
parameters:
  csi.storage.k8s.io/fstype: ext4
  type: gp3 # General Purpose SSD
```

DKP deploys with gp3 (general purpose SSDs) EBS volumes.

- Driver documentation: aws-ebs-csi-driver[82]
- Volume types and pricing: volume types[83]

3.4.2.2.2 Azure CSI Driver

DKP deploys with StandardSSD_LRS for Azure Virtual Disks.

- Driver documentation: azuredisk-csi-driver[84]
- Volume types and pricing: volume types[85]
- Specifics for Azure using Pre-provisioning can be found here: Pre-provisioned Azure only Configurations (see page 1079)

3.4.2.2.3 vSphere CSI Driver

DKP default storage class for vSphere supports dynamic provisioning and static provisioning of block volumes.

- Driver documentation: https://docs.vmware.com/en/VMware-vSphere-Container-Storage-Plug-in/index.html
- Specifics for using vSphere storage driver: https://docs.vmware.com/en/VMware-vSphere-Container-Storage-Plug-in/2.0/vmware-vsphere-csp-getting-started/GUID-5D144DA0-4806-4DEB-8819-10A1C42E38AB.html

3.4.2.2.4 Cloud Director

In VMware Cloud Director:

- The Cloud provider component (CPI)[86] purpose is to manage the lifecycle of the load balancers as well as associate Kubernetes nodes with virtual machines in the infrastructure.
- CAPVCD is a component that runs in a Kubernetes cluster that connects to the VCD API. It uses the CPI to do the work of creating and managing the infrastructure.
- For storage, DKP has a CSI plugin for interfacing with CPI and can create disks dynamically and becomes the StorageClass

82 https://github.com/kubernetes-sigs/aws-ebs-csi-driver
83 https://aws.amazon.com/ebs/features/
84 https://github.com/kubernetes-sigs/azuredisk-csi-driver
85 https://learn.microsoft.com/en-us/azure/aks/azure-disk-csi
86 https://github.com/vmware/cloud-provider-for-cloud-director/

3.4.2.2.5 Pre-provisioned CSI Driver

In a pre-provisioned (see page 84) environment, DKP will also deploy a CSI compatible driver and configure a default StorageClass - `localvolumeprovisioner`.

- Driver documentation: local-static-provisioner[87]

> DKP uses local-static-provisioner[88] (`localvolumeprovisioner`) as the default storage provider for a pre-provisioned environment. However, local-static-provisioner[89] is not suitable for production use. You should use a different Kubernetes CSI[90] compatible storage that is suitable for production.

To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass by following the steps in the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/. You can choose from any of the storage options available for Kubernetes and make your storage choice[91] the default storage.

Ceph can also be used as CSI storage. Refer to that section in the documentation for information on how to use Rook Ceph (see page 1041).

3.4.2.2.6 GCP CSI Driver

This driver allows volumes backed by Google Cloud Filestore instances to be dynamically created and mounted by workloads.

- Driver documentation: gcp-filestore-csi-driver[92]
- Persistent volumes and dynamic provisioning: volume types[93]

3.4.2.3 Related Information

- Kubernetes Storage[94]
- Kubernetes CSI Storage Drivers[95]

87 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner
88 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner
89 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner
90 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
91 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
92 https://github.com/kubernetes-sigs/gcp-filestore-csi-driver
93 https://cloud.google.com/kubernetes-engine/docs/concepts/persistent-volumes
94 https://kubernetes.io/docs/concepts/storage/
95 https://kubernetes-csi.github.io/docs/drivers.html

- Kubernetes Local Persistent Volumes[96]
- Persistent Volumes[97]
- DKP vSphere Storage (see page 445)

3.4.3 Provision a Static Local Volume

Learn how to provision a static local volume for a DKP cluster

> ⚠ When creating a pre-provisioned infrastructure cluster, DKP uses `localvolumeprovisioner` as the default storage provider (see page 110). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[98] compatible storage that is suitable for production.

You can choose from any of the storage options available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly-created StorageClass by following the steps in the Kubernetes documentation, Changing the Default Storage Class[99].

The `localvolumeprovisioner` component uses the local volume static provisioner[100] to manage persistent volumes for pre-allocated disks. It does this by watching the `/mnt/disks` folder on each host and creating persistent volumes in the `localvolumeprovisioner` storage class for each disk that it discovers in this folder.

- Persistent volumes with a 'Filesystem' volume-mode are discovered if you mount them under `/mnt/disks`.
- Persistent volumes with a 'Block' volume-mode are discovered if you create a symbolic link to the block device in `/mnt/disks`.

For additional DKP documentation regarding StorageClass, see: Default Storage Providers in DKP (see page 110)

3.4.3.1 Before you Begin

Before starting, verify the following:

- You have access to a Linux, macOS, or Windows computer with a supported operating system version.

96 https://kubernetes.io/blog/2019/04/04/kubernetes-1.14-local-persistent-volumes-ga/
97 https://kubernetes.io/docs/concepts/storage/persistent-volumes/#class-1
98 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
99 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/
100 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner

- You have a provisioned `dkp` cluster that uses the `localvolumeprovisioner` platform application, but have not added any other Kommander applications to the cluster yet.

This distinction between provisioning and deployment is important because some applications depend on the storage class provided by the `localvolumeprovisioner` component and can fail to start if it is not configured yet.

3.4.3.2 Provision the Cluster and a Volume

1. Create a pre-provisioned cluster by following the steps outlined in the Pre-provisioned Infrastructure (see page 1070) topic.

   As volumes are created/mounted on the nodes, the local volume provisioner detects each volume in the `/mnt/disks` directory and adds it as a persistent volume with the `localvolumeprovisioner` storage class. For more information, see the documentation regarding Kubernetes Local Storage[101].

2. Create at least one volume in `/mnt/disks` on each host.

   For example, mount a `tmpfs` volume:

   ```
   mkdir -p /mnt/disks/example-volume && mount -t tmpfs example-volume /mnt/disks/
   example-volume
   ```

3. Verify the persistent volume by running the following command:

   ```
   kubectl get pv
   ```

   The command displays output similar to the following:

   ```
   NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
   CLAIM   STORAGECLASS              REASON   AGE
   local-pv-4c7fc8ba   3986Mi     RWO            Delete           Available
   localvolumeprovisioner            2s
   ```

4. Claim the persistent volume using a PersistentVolumeClaim, by running the following command:

   ```
   cat <<EOF | kubectl create -f -
   kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: example-claim
   spec:
   ```

101 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner/blob/master/docs/operations.md#link-devices-into-directory-to-be-discovered-as-block-pvs

```
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
  storageClassName: localvolumeprovisioner
EOF
```

5. Reference the persistent volume claim in a pod by running the following command:

```
cat <<EOF | kubectl create -f -
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-persistent-volume
spec:
  containers:
    - name: frontend
      image: nginx
      volumeMounts:
        - name: data
          mountPath: "/var/www/html"
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: example-claim
EOF
```

6. Verify the persistent volume claim by running the following command:

```
kubectl get pvc
```

The command displays output similar to the following:

```
NAME            STATUS   VOLUME              CAPACITY   ACCESS MODES
STORAGECLASS             AGE
example-claim   Bound    local-pv-4c7fc8ba   3986Mi     RWO
localvolumeprovisioner   78s
```

And you can also check the persistent volume:

```
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM                   STORAGECLASS             REASON   AGE
local-pv-4c7fc8ba   3986Mi     RWO            Delete           Bound
default/example-claim   localvolumeprovisioner            15m
```

Upon deletion of the persistent volume claim, the corresponding persistent volume resource uses the *Delete* reclaim policy, which removes all data on the volume.

3.5 Resource Requirements

To ensure a successful DKP install, there are certain requirements to be met in regards to Control Plan nodes as well as Worker nodes. These resource requirements can be slightly different for different Infrastructure Providers.

- Managed Cluster Requirements (see page 120)
- Management Cluster Application Requirements (see page 122)
- Workspace Platform Application Defaults and Resource Requirements (see page 124)

The specific infrastructure provider resources are listed in their install sections which are referenced below.

3.5.1 General Resource Requirements

To Install DKP (see page 127) with the minimum amount of resources, please see the list below before beginning installation. You need at least three control plane nodes and you must have at least four worker nodes. The specific number of worker nodes required for your environment can vary depending on the cluster workload and size of the nodes.

	Control Plane Nodes	**Worker Nodes**
vCPU Count	4	8
Memory	16 GB	32 GB
Disk Volume	Approximately 80 GiB • used for `/var/lib/kubelet` and `/var/lib/containerd`	Approximately 95 GiB • used for `/var/lib/kubelet` and `/var/lib/containerd`
Root Volume	Disk usage must be below 85%	Disk usage must be below 85%

If you use the instructions to create a cluster using the DKP default settings without any edits to configuration files or additional flags, your cluster is deployed on an Ubuntu 20.04 operating system image (see page 67) with 3 control plane nodes, and 4 worker nodes which match the requirements above.

3.5.2 Infrastructure Provider Specific

For Specific Infrastructure Providers additional requirements may apply. For example, DKP on Azure defaults to deploying a `Standard_D4s_v3` virtual machine with an 128 GiB volume for the OS and an 80GiB volume for etcd storage, which meets the above requirements. See Main Page for Provider for infrastructure provider specific additional resource information:

- Pre-provisioned Install Options (see page 148)
- AWS Install Options (see page 294)
- Azure Install Options (see page 541)
- vSphere Install Options (see page 441)
- VMware Cloud Director Install Options (see page 538)

3.5.3 Kommander Component Requirements:

- Management Cluster Application Requirements (see page 122)
- Workspace Platform Application Defaults and Resource Requirements (see page 124)

3.5.3.1 Next Topic:

Install Overview (see page 127)

3.5.4 Managed Cluster Requirements

To create additional clusters that will be managed by your Management cluster (see page 79), please ensure you have at least the minimal recommendation of required resources below.

3.5.4.1 Minimum Recommendation for Managed Clusters:

Worker Nodes - For a default installation, at least four worker nodes with:

- 8 CPU cores each
- 12Gi of memory
- A storage class and disk mounts that can accommodate at least four persistent volumes

NOTE: Four worker nodes are required in order to support upgrades to the `rook-ceph` platform application. `rook-ceph` is used to support the logging stack, the `velero` backup tool, and DKP Insights. If you have disabled the `rook-ceph` platform application, only three worker nodes are required.

Control Plane Nodes:

- 8 CPU cores each
- 12Gi of memory

NOTE: One control plane node is acceptable for a non-critical test environment, but any production workload must have at least three control planes.

The cluster needs:

- default Storage Slass and four volumes of 32G, 32G, 2G, and 100G or the ability to create those volumes depending on the Storage Class:

```
$ kubectl get pv -A
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS   CLAIM
STORAGECLASS   REASON   AGE
pvc-08de8c06-bd66-40a3-9dd4-b0aece8ccbe8   32Gi       RWO            Delete
Bound    kommander-default-workspace/kubecost-cost-analyzer
ebs-sc                  124m
pvc-64552486-7f4c-476a-a35d-19432b3931af   32Gi       RWO            Delete
Bound    kommander-default-workspace/kubecost-prometheus-server
ebs-sc                  124m
pvc-972c3ee3-20bd-449b-84d9-25b7a06a6630   2Gi        RWO            Delete
Bound    kommander-default-workspace/kubecost-prometheus-alertmanager
ebs-sc                  124m
pvc-98ab93f1-2c2f-46b6-b7d3-505c55437fbb   100Gi      RWO            Delete
Bound    kommander-default-workspace/db-prometheus-kube-prometheus-stack-prometheus-0
  ebs-sc                  123m
```

NOTE: Actual workloads may demand more resources depending on usage.

3.5.5 Management Cluster Application Requirements

3.5.5.1 Management cluster application minimum resources and storage requirements

This section only details requirements for management cluster-specific applications in the Kommander component. For the list of all platform applications, see Platform Application Configuration Requirements (see page 662).

This table describes the workspace platform applications specific to the management cluster, minimum resource requirements, minimum persistent storage requirements, and default priority class values.

Common Name	App ID (for App versions, see the Release Notes[102])	Deployed by default	Minimum Resources Suggested	Minimum Persistent Storage Required	Default Priority Class
Centralized Grafana*	centralized-grafana	Yes	cpu: 200m memory: 100Mi		DKP Critical (100002000)
Centralized Kubecost*	centralized-kubecost	Yes	cpu: 1200m memory: 4151Mi	# of PVs: 1 PV sizes: 32Gi	DKP High (100001000)
Chartmuseum	chartmuseum	Yes		# of PVs: 1 PV sizes: 2Gi	DKP Critical (100002000)
Dex	dex	Yes	cpu: 100m memory: 50Mi		DKP Critical (100002000)
Dex Authenticator	dex-k8s-authenticator	Yes	cpu: 100m memory: 128Mi		DKP High (100001000)
DKP Insights Management	dkp-insights-management	No	cpu: 100m memory: 128Mi		DKP Critical (100002000)

102 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/110723415/DKP+2.4.0+Components+and+Applications#applications

Common Name	App ID (for App versions, see the Release Notes)	Deployed by default	Minimum Resources Suggested	Minimum Persistent Storage Required	Default Priority Class
Gitea	gitea	Yes	cpu: 500m memory: 512Mi	# of PVs: 2 PV sizes: 10Gi	DKP Critical (100002000)
Karma*	karma	Yes			DKP Critical (100002000)
Kommander	kommander	No	cpu: 1100m memory: 896Mi		DKP Critical (100002000)
Kommander AppManagement	kommander-appmanagement	Yes	cpu: 300m memory: 256Mi		DKP Critical (100002000)
Kommander Flux	kommander-flux	Yes	cpu: 5000m memory: 5Gi		DKP Critical (100002000)
Kommander UI	kommander-ui	No	cpu: 100m memory: 256Mi		DKP Critical (100002000)
Kubefed	kubefed	Yes	cpu: 300m memory: 192Mi		DKP Critical (100002000)
Kubetunnel	kubetunnel	Yes	cpu: 200m memory: 148Mi		DKP Critical (100002000)
Thanos*	thanos	Yes			DKP Critical (100002000)
Traefik ForwardAuth	traefik-forward-auth-mgmt	Yes	cpu: 100m memory: 128Mi		DKP Critical (100002000)

Applications designated with an asterisk (“*”) are DKP Enterprise only apps that are deployed by default for DKP Enterprise Customers only.

3.5.6 Workspace Platform Application Defaults and Resource Requirements

The following table provides a list of platform applications that are available in DKP with the Kommander component. Some of them are deployed by default on attachment, others require manual installation (see page 667).

Workspace platform applications require more resources than solely deploying or attaching clusters into a workspace. Your cluster must have sufficient resources when deploying or attaching to ensure that the platform services are installed successfully.

The following table describes all the workspace platform applications that are available to the clusters in a workspace, minimum resource requirements, whether they are enabled by default, and their default priority classes.

Common Name	App ID (for App versions, see the Release Notes[103])	Deployed by default	Minimum Resources Suggested	Minimum Persistent Storage Required	Default Priority Class
Cert Manager	cert-manager	Yes	cpu: 10m memory: 32Mi		system-cluster-critical (2000000000)
External DNS	external-dns	No			DKP High (100001000)
Fluent Bit	fluent-bit	No	cpu: 350m memory: 350Mi		DKP Critical (100002000)
Gatekeeper	gatekeeper	Yes	cpu: 300m memory: 768Mi		system-cluster-critical (2000000000)
Grafana	grafana-logging	No	cpu: 200m memory: 100Mi		DKP Critical (100002000)

103 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273745953/2.6+DKP+2.6.0+Components+and+Applications

Common Name	App ID (for App versions, see the Release Notes)	Deployed by default	Minimum Resources Suggested	Minimum Persistent Storage Required	Default Priority Class
Loki	grafana-loki	No		# of PVs: 8 PV sizes: 10Gi x 8 (total: 80Gi)	DKP Critical (100002000)
Istio	istio	No	cpu: 1270m memory: 4500Mi		DKP Critical (100002000)
Jaeger	jaeger	No			DKP High (100001000)
Kiali	kiali	No	cpu: 20m memory: 128Mi		DKP High (100001000)
Knative	knative	No	cpu: 610m memory: 400Mi		DKP High (100001000)
Kube OIDC Proxy	kube-oidc-proxy	Yes			DKP Critical (100002000)
Kube Prometheus Stack	kube-prometheus-stack	Yes	cpu: 1300m memory: 4300Mi	# of PVs: 1 PV sizes: 100Gi	DKP Critical (100002000)
Kubecost	kubecost	Yes	cpu: 700m memory: 1700Mi	# of PVs: 3 PV sizes: 2Gi, 32Gi, 32Gi (total: 66Gi)	DKP High (100001000)
Kubernetes Dashboard	kubernetes-dashboard	Yes	cpu: 250m memory: 300Mi		DKP High (100001000)

Common Name	App ID (for App versions, see the Release Notes)	Deployed by default	Minimum Resources Suggested	Minimum Persistent Storage Required	Default Priority Class
Logging Operator	logging-operator	No	cpu: 350m * # of nodes + 600m memory: 228Mi + 350Mi * # of nodes	# of PVs: 1 PV sizes: 10Gi	DKP Critical (100002000)
NFS Server Provisioner	nfs-server-provisioner	No	# of PVs: 1 PV sizes: 100Gi		DKP High (100001000)
NVIDIA GPU Operator	nvidia-gpu-operator	No	cpu: 100m memory: 128Mi		system-cluster-critical (2000000000)
Prometheus Adapter	prometheus-adapter	Yes	cpu: 1000m memory: 1000Mi		DKP Critical (100002000)
Reloader	reloader	Yes	cpu: 100m memory: 128Mi		DKP High (100001000)
Rook Ceph	rook-ceph	Yes	cpu: 100m memory: 128Mi		system-cluster-critical (2000000000)
Rook Ceph Cluster	rook-ceph-cluster	Yes	cpu 2500m mem 8Gi	# of PVs: 4 PV sizes: 40Gi	DKP Critical (100002000) system-cluster-critical (2000000000) system-node-critical
Traefik	traefik	Yes	cpu: 500m		DKP Critical (100002000)

Common Name	App ID (for App versions, see the Release Notes)	Deployed by default	Minimum Resources Suggested	Minimum Persistent Storage Required	Default Priority Class
Traefik ForwardAuth	traefik-forward-auth	Yes	cpu: 100m memory: 128Mi		DKP Critical (100002000)
Velero	velero	No	cpu: 1000m memory: 1024Mi		DKP Critical (100002000)

- Currently, DKP only supports a single deployment of `cert-manager` per cluster. Because of this, `cert-manager` cannot be installed on any Konvoy managed clusters or clusters that come with `cert-manager` pre-installed.
- Only a single deployment of `traefik` per cluster is supported.
- DKP automatically manages the deployment of `traefik-forward-auth` and `kube-oidc-proxy` when clusters are attached to the workspace. These applications are not shown in the DKP UI.
- Applications are enabled in DKP and then deployed to attached clusters. To confirm that your enabled application has successfully deployed, you should verify via the CLI (see page 700).

3.5.6.1 Next Topic:

Install Overview (see page 127)

3.6 Install Overview

3.6.1 Prepare Your Environment for Install:

Follow the steps below to install the basic package requirements for your environment to achieve a successful install of DKP. Next install DKP, and then you can begin any custom configurations based on your environment.

1. Install required packages. In most cases, you can install the required software using your preferred package manager. For example, on a macOS computer, you can use Homebrew[104] to install `kubectl` and the `aws` command-line utility by running the following command:

   ```
   brew install kubernetes-cli awscli
   ```

2. Check the Kubernetes client version. Many important Kubernetes functions **do not work** if your client is outdated. You can verify that the version of `kubectl` you have installed is supported by running the following command:

   ```
   kubectl version --short=true
   ```

3. Check supported Kubernetes versions (see page 139) after finding your version with the command above.
4. For air-gapped, create a bastion host (see page 1068) for the cluster nodes to use within the air-gapped network. This bastion host needs access to a local registry in lieu of an Internet connection for pulling images. The recommended template naming pattern is `../folder-name/dkp-e2e-bastion-template` or similar. Each infrastructure provider has its own set of bastion host instructions. Refer to your provider's site for details - Azure[105], AWS[106], GCP[107], or vSphere[108].
5. For creating DKP machine images, you need to download Konvoy Image Builder (see page 1626) and extract it.
6. To download DKP, see the Download (see page 76) topic for information. You will need to download and extract the DKP binary package tarball.
7. Verify you have valid **cloud provider security credentials** to deploy the cluster on that platform.

> This step regarding provider security credentials is not required if you are installing DKP on an on-premises environment. For information about installing in an on-premises environment, see Install Pre-provisioned (see page 1070).

8. Install Konvoy depending on which infrastructure you have. Consult the Day 1 - Basic Installs by Infrastructure (see page 146) for provider specific installs. To use customized YAML and other advanced features, consult the Custom Installation and Additional Infrastructure Tools (see page 1050)

104 https://docs.brew.sh/Installation
105 https://learn.microsoft.com/en-us/azure/bastion/quickstart-host-portal
106 https://aws.amazon.com/solutions/implementations/linux-bastion/
107 https://blogs.vmware.com/cloud/2021/06/02/intro-google-cloud-vmware-engine-bastion-host-access-iap/
108 https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.security.doc/GUID-6975426F-56D0-4FE2-8A58-580B40D2F667.html

section of the documentation for steps on creating a customized cluster with the Konvoy component of DKP using YAML. Find your infrastructure provider listed in that section.

9. Now that you have a basic Kubernetes cluster installed through Konvoy, configure the Kommander (see page 1530) component.
10. Lastly, continue with initializing the configuration file under the Kommander Installer Configuration File (see page 1558) component of DKP.

You may also want to test operations by deploying a simple, sample application, customizing the cluster configuration, or checking the status of cluster components.

3.6.2 Next Step:

Prerequisites for Install (see page 141)

3.6.3 Install OS Version Compatibility

Below is a list of Supported Operating Systems and the versions for the latest release.

3.6.3.1 Amazon Web Services (AWS)

Operating System	Kernel	Default Config (see page 295)	FIPS (see page 333)	Air Gapped (see page 312)	FIPS with Air Gapped (see page 350)	GPU Support (see page 369)	GPU with Air Gapped (see page 389)	Konvoy Image Builder (see page 1626)
CentOS 7.9[109]	3.10.0-1160.80.1.el7.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 7.9[110]	3.10.0-1160.80.1.el7.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 8.4[111]	4.18.0-305.88.1.el8_4.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes

109 https://wiki.centos.org/Manuals(2f)ReleaseNotes(2f)CentOS7(2e)2009.html
110 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/7.9_release_notes/index
111 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.4_release_notes/index

Operating System	Kernel	Default Config (see page 295)	FIPS (see page 333)	Air Gapped (see page 312)	FIPS with Air Gapped (see page 350)	GPU Support (see page 369)	GPU with Air Gapped (see page 389)	Konvoy Image Builder (see page 1626)
RHEL 8.6[112]	4.18.0-372.70.1.el8_6.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 8.8[113]	4.18.0-477.27.1.el8_8.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Ubuntu 18.04 (Bionic Beaver)[114]	5.4.0-1103-aws	Yes						Yes
Ubuntu 20.04 (Focal Fossa)[115]	5.15.0-1051-aws	Yes				Yes		Yes
SLES 15 SP3[116]	5.14.21-150500.55.7-default	Yes				Yes		Yes
Oracle Linux RHCK 7.9[117]	3.10.0-1160.105.1.0.1.el7.x86_64	Yes	Yes					Yes
Rocky Linux 9.1[118]	5.14.0-162.12.1.el9_1.0.2.x86_64	Yes		Yes				Yes
Flatcar 3033.3.x[119]	5.10.198-flatcar	Yes						Yes

112 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/8.6_release_notes/index
113 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.8_release_notes/index
114 https://wiki.ubuntu.com/BionicBeaver/ReleaseNotes
115 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes
116 https://documentation.suse.com/en-us/sles/15-SP3/
117 https://docs.oracle.com/en/operating-systems/oracle-linux/7/relnotes7.9/
118 https://docs.rockylinux.org/release_notes/9_1/
119 https://www.flatcar.org/releases/#lts-release

3.6.3.2 Microsoft Azure

Operating System	Kernel	Default Config (see page 541)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1642)
Ubuntu 20.04 (Focal Fossa)[120]	5.15.0-1053-azure	Yes						Yes
Rocky Linux 9.1[121]	5.14.0-162.12.1.el9_1.0.2.x86_64	Yes						Yes

3.6.3.3 Google Cloud Platform (GCP)

Operating System	Kernel	Default Config (see page 572)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1646)
CentOS 7.9[122]	3.10.0-1160.62.1.el7.x86_64	Yes						Yes
Ubuntu 18.04[123]	5.4.0-1072-gcp	Yes						Yes
Ubuntu 20.04[124]	5.13.0-1024-gcp	Yes						Yes

120 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes
121 https://docs.rockylinux.org/release_notes/9_0/
122 https://wiki.centos.org/action/show/Manuals/ReleaseNotes/CentOS7.2003
123 https://wiki.ubuntu.com/BionicBeaver/ReleaseNotes
124 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes

3.6.3.4 Pre-Provisioned

Pre-provisioned (see page 148) includes on-premises, vSphere, AWS, Azure, and GCP infrastructures and is described in more detail under What is Pre-provisioned Infrastructure (see page 84).

Operating System	**Kernel**	Default Config (see page 152)	FIPS (see page 194)	Air Gapped (see page 169)	FIPS with Air-Gapped (see page 214)	GPU Support (see page 239)	GPU with Air Gapped (see page 262)	Konvoy Image Builder (see page 1660)
CentOS 7.9[125]	3.10.0-1160.80.1.el7.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 7.9[126]	3.10.0-1160.80.1.el7.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 8.4[127]	4.18.0-305.88.1.el8_4.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 8.6[128]	4.18.0-372.70.1.el8_6.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
RHEL 8.8[129]	4.18.0-477.27.1.el8_8.x86_64	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Flatcar 3033.3.x[130]	3033.3.16-flatcar	Yes						Yes

125 https://wiki.centos.org/action/show/Manuals/ReleaseNotes/CentOS7.2003
126 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/7.9_release_notes/index
127 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.4_release_notes/index
128 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/8.6_release_notes/index
129 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.8_release_notes/index
130 https://www.flatcar.org/releases/#lts-release

Operating System	**Kernel**	Default Config (see page 149)	FIPS (see page 194)	Air Gapped (see page 169)	FIPS with Air-Gapped (see page 214)	GPU Support (see page 239)	GPU with Air Gapped (see page 262)	Konvoy Image Builder (see page 1660)
Ubuntu 20.04[131]	5.15.0-1048-aws	Yes				Yes		Yes
SLES 15 SP3[132]	5.3.18-59.37-default	Yes				Yes		Yes
Oracle Linux RHCK 7.9[133]	5.4.17-2136.314.6.3.el7uek.x86_64	Yes						Yes
Rocky Linux 9.1[134]	5.14.0-162.12.1.el9_1.0.2.x86_64	Yes		Yes				Yes

131 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes
132 https://documentation.suse.com/en-us/sles/15-SP3/
133 https://docs.oracle.com/en/operating-systems/oracle-linux/7/relnotes7.9/
134 https://docs.rockylinux.org/release_notes/9_1/

3.6.3.5 Pre-provisioned/Azure

Operating System	Kernel	Default Config[135][136]	FIPS[137][138]	Air Gapped[139][140]	FIPS with Air Gapped[141][142]	GPU Support[143][144]	GPU with Air Gapped[145][146]	Konvoy Image Builder[147][148]
RHEL 8.6[149]	4.18.0-372.52.1.el8_6.x86_64	Yes	Yes	Yes	Yes			Yes

3.6.3.6 vSphere

vCenter Server + ESXi

Operating System	Kernel	Default Config (see page 447)	FIPS (see page 492)	Air Gapped (see page 468)	FIPS with Air Gapped (see page 514)	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1652)
RHEL 7.9[150]	3.10.0-1160.el7.x86_64	Yes	Yes	Yes	Yes			Yes

135 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273320894/2.6%2BPre-provisioned%2BNon-air-gapped%2BInstall
136 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273320894/2.6%2BPre-provisioned%2BNon-air-gapped%2BInstall
137 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273385679/2.6%2BPre-provisioned%2BFIPS%2BInstall
138 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273385679/2.6%2BPre-provisioned%2BFIPS%2BInstall
139 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273321203/2.6%2BPre-provisioned%2BAir-gapped%2BInstall
140 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273321203/2.6%2BPre-provisioned%2BAir-gapped%2BInstall
141 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273647809/2.6+Pre-provisioned+FIPS+Air-gapped+Install
142 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273647809/2.6+Pre-provisioned+FIPS+Air-gapped+Install
143 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273387183/2.6+Pre-provisioned+with+GPU+Install
144 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273387183/2.6+Pre-provisioned+with+GPU+Install
145 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273319903/2.6+Pre-provisioned+Air-gapped+with+GPU+Install
146 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273319903/2.6+Pre-provisioned+Air-gapped+with+GPU+Install
147 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273713682/2.6+KIB+with+Pre-provisioned+Environments
148 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/273713682/2.6+KIB+with+Pre-provisioned+Environments
149 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/8.6_release_notes/index
150 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/7.9_release_notes/index

Operating System	Kernel	Default Config (see page 447)	FIPS (see page 492)	Air Gapped (see page 468)	FIPS with Air Gapped (see page 514)	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1652)
RHEL 8.4[151]	4.18.0-305.el8.x86_64	Yes	Yes	Yes	Yes			Yes
RHEL 8.6[152]	4.18.0-372.9.1.el8.x86_64	Yes	Yes	Yes	Yes			Yes
RHEL 8.8[153]	4.18.0-477.10.1.el8_8.x86_64	Yes	Yes	Yes	Yes			Yes
Ubuntu 20.04[154]	5.4.0-125-generic	Yes		Yes				Yes
Rocky Linux 9.1[155]	5.14.0-162.6.1.el9_1.x86_64	Yes		Yes				Yes
Flatcar 3033.3.x[156]	3033.3.16-flatcar	Yes						Yes

151 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.4_release_notes/index
152 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/8.6_release_notes/index
153 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.8_release_notes/index
154 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes
155 https://docs.rockylinux.org/release_notes/9_1/
156 https://www.flatcar.org/releases/#lts-release

3.6.3.7 VMware Cloud Director (VCD)

Operating System	Kernel	Default Config (see page 1444)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1652)
Ubuntu 20.04[157]	5.4.0-125-generic	Yes						Yes

3.6.3.8 Amazon Elastic Kubernetes (EKS)

Operating System	Kernel	Default Config (see page 413)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1642)
Amazon Linux 2 v7.9[158]	5.10.199-190.747.amzn2.x86_64	Yes						

157 https://wiki.ubuntu.com/FocalFossa/ReleaseNotes
158 https://docs.aws.amazon.com/AL2/latest/relnotes/relnotes-al2.html

3.6.3.9 Azure Kubernetes Service (AKS)

Operating System	Kernel	Default Config (see page 556)	FIPS	Air Gapped	FIPS with Air Gapped	GPU Support	GPU with Air Gapped	Konvoy Image Builder (see page 1646)
Ubuntu 22.04.2 LTS[159]	5.15.0-1051-azure	Yes						

3.6.4 Install Related Version Information

Other helpful links for versions required to successfully run with DKP for the latest version can be found below and also in their respective section of Documentation.

3.6.4.1 Supported Kubernetes Version

3.6.4.2 Deploying Clusters Versions

The latest supported Kubernetes versions for DKP are listed below. In DKP 2.8.1, the Kubernetes version installed by default is 1.28.7. The newest and oldest Kubernetes versions used with DKP must be within one minor version.

- Latest supported Kubernetes version is at **1.28.7 for deploying clusters**
- Other Kubernetes versions are supported at **1.27.9 for deploying clusters in EKS**

Kubernetes 1.28.x enables you to benefit from the latest features and security fixes in upstream Kubernetes. This release comes with 60 enhancements that you can benefit from such as Node log access via Kubernetes API[160], Seccomp profile defaulting, and much more.

To read more about major features in this release, visit this page[161] and https://kubernetes.io/blog/2023/04/11/kubernetes-v1-27-release/ .

3.6.4.3 Attaching Clusters Versions

DKP 2.8.1 supports attaching clusters with the following Kubernetes versions:

159 https://discourse.ubuntu.com/t/jammy-jellyfish-release-notes/24668
160 https://github.com/kubernetes/enhancements/issues/2258
161 https://github.com/kubernetes/sig-release/blob/master/releases/release-1.27/README.md

Product	Compatible Kubernetes Versions
EKS	1.27.9
AKS	1.28.x
GKE	1.27.x

Attaching Kubernetes clusters with versions greater than n-1 is not recommended.

When attempting to attach a cluster with a lower Kubernetes version than the DKP default, you need to build and use an image with that lower version of Kubernetes. See Konvoy Image Builder (see page 1626) for documentation on building images. Failure to do this could result in an error.

3.6.4.4 Konvoy Image Builder

Refer to the main page (see page 1626)for a table of components versions and compatibility.

3.6.4.5 Storage Providers

Default storage providers in DKP

When deploying DKP using a supported cloud provisioner (AWS, Azure, GCP), DKP automatically configures native storage drivers for the target platform. In addition, DKP deploys a default StorageClass[162] for https://kubernetes.io/docs/concepts/storage/dynamic-provisioning/ creation. The table below lists the driver and default StorageClass for each supported cloud provisioner.

162 https://kubernetes.io/docs/concepts/storage/storage-classes/

Cloud Provisioner	Version	Driver	Default Storage Class
AWS	`1.23` The AWS CSI driver is upgraded to a new minor version in DKP 2.7, which has an upgraded version[163] of the package.	aws-ebs-csi-driver[164]	`ebs-sc`
Azure	`1.29.0`	azuredisk-csi-driver[165]	`azuredisk-sc`
Pre-provisioned	`2.6.0`	local-static-provisioner[166]	`localvolumeprovisioner`
vSphere	`2.7.2`	vsphere-csi-driver[167]	`vsphere-raw-block-sc`
Cloud Director (VCD)	`1.4.0-d2iq.0`	cloud-director-named-csi-driver[168]	`vcd-disk-sc`

163 https://github.com/kubernetes-sigs/aws-ebs-csi-driver/blob/master/CHANGELOG.md#urgent-upgrade-notes
164 https://github.com/kubernetes-sigs/aws-ebs-csi-driver
165 https://github.com/kubernetes-sigs/azuredisk-csi-driver
166 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner
167 https://github.com/kubernetes-sigs/vsphere-csi-driver
168 https://github.com/vmware/cloud-director-named-disk-csi-driver

Cloud Provisioner	Version	Driver	Default Storage Class
GCP	`1.10.3`	gcp-compute-persistent-disk-csi-driver[169]	`csi-gce-pd`

DKP uses the local static provisioner as the default storage provider for Pre-provisioned clusters. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[170] compatible storage that is suitable for production.
You can choose from any of the storage options[171] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[172].

3.6.4.6 Deploying a Cluster in FIPS mode

In order to create a cluster in FIPS mode, we must inform the bootstrap controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes.

3.6.4.6.1 Supported FIPS Builds

Component	Repository	Version
Kubernetes	docker.io/mesosphere[173]	v1.28.7+fips.0
etcd	docker.io/mesosphere[174]	3.5.10+fips.0

169 https://github.com/kubernetes-sigs/gcp-compute-persistent-disk-csi-driver/releases
170 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
171 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
172 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/
173 https://hub.docker.com/layers/mesosphere/kube-apiserver/v1.28.7_d2iq.0/images/sha256-1cafe40f5a17c86aa75574b25dd637bbeb377e2ac703532d72f1118c1e966e28?context=explore
174 https://hub.docker.com/layers/mesosphere/etcd/v3.5.10_fips.0/images/sha256-60da8d0d3b37e71a4fa918ffa7ffd9963d9892b3ba43b8f63119d806f2e585ed?context=explore

3.7 Prerequisites for Install

Before you create a DKP image and deploy the initial DKP cluster, the operator's machine is required to be either an OSX or Linux-based machine of a supported version. Ensure you have all the prerequisites met for both the Konvoy and Kommander component.

Additional prerequisites are necessary for **air-gapped**, so verify that you have all non-air-gapped met and then any additional air-gapped prerequisites listed.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)

3.7.1 Prerequisites for the Konvoy Component

For DKP and Konvoy Image Builder to run, the operator machine requires:

3.7.1.1 Non-air-gapped (all environments)

In a non-air-gapped environment, your environment has two-way access to and from the **Internet**. Below are the prerequisites required if installing in a non-air-gapped environment:

- An x86_64-based Linux or macOS machine.
- The `dkp` binary on the bastion by downloading (see page 76). To check which version of DKP you installed for compatibility reasons, run the `dkp version` command (dkp version (see page 1943)).
- A Container engine/runtime installed is required to install DKP:
  - Version Docker®[175] container engine version 18.09.2 or higher installed for Linux or MacOS - On macOS, Docker runs in a virtual machine which needs configured with at least 8 GB of memory.
  - Version 4.0 of Podman[176] or higher for Linux. Host requirements found here: Host Requirements[177]
- kubectl[178] for interacting with the running cluster on the host where the DKP Konvoy command line interface (CLI) runs.
- Konvoy Image Builder (see page 1626) (KIB)
- Valid Provider account with credentials configured.

175 https://docs.docker.com/get-docker/
176 https://podman.io/getting-started/installation
177 https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
178 https://kubernetes.io/docs/tasks/tools/#kubectl

    - AWS credentials configured[179] that can manage CloudFormation Stacks, IAM Policies, IAM Roles, and IAM Instance Profiles
    - Azure credentials configured[180]
- CLI tooling of the cloud provider being used to deploy DKP commands:
    - aws-cli[181]
    - googlecloud-cli[182]
    - azure-cli[183]
- **EKS and AKS only**: A management cluster (self-managed) cluster is required:
    - If you follow Day 1 - Basic Installation (see page 146) instructions for install, you use the `--self-managed` flag and therefore have a self-managed cluster. If you use the instructions under Custom Installation and Additional Infrastructure Tools (see page 1050), ensure you perform the self-managed process on your new cluster:
    - A Self-managed AWS Cluster (see page 1227)
    - A Self-managed Azure Cluster (see page 1324)
- **Pre-provisioned on Premises only**:
    - Pre-provisioned hosts with SSH access enabled.
    - An unencrypted SSH private key, whose public key is configured on the above hosts.
    - Pre-provisioned Override Files (see page 1680) if needed.
- **vSphere only**:
    - A valid VMware vSphere account with credentials configured.

DKP requires permissions for the cloud provider that is being used. See the Supported Infrastructure Operating Systems (see page 67) page for further setup requirements.

3.7.1.2 Air-gapped Only (**additional** prerequisites)

In an air-gapped environment, your environment is **isolated** from unsecured networks, like the Internet, and therefore require additional considerations for installation. Below are the **additional** prerequisites required if installing in an air-gapped environment:

- Linux machine (bastion) that has access to the existing VPC. (Instead of An x86_64-based Linux or macOS machine).

179 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html
180 https://github.com/kubernetes-sigs/cluster-api-provider-azure/blob/master/docs/book/src/topics/getting-started.md#prerequisites
181 https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html
182 https://cloud.google.com/sdk/docs/install
183 https://learn.microsoft.com/en-us/cli/azure/install-azure-cli

- Ability to download artifacts from the internet and then copy those onto your bastion machine.
- Download (see page 76) the Complete DKP Air-gapped Bundle for this release - `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` .
    - To use a local registry whether air-gapped or non-air-gapped environment, download and extract the the complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`) to load your registry.
- An existing local registry (see page 1606) to seed the air-gapped environment.

3.7.2 Prerequisites for Kommander Component

To install the Kommander component of DKP, ensure you meet the following prerequisites:

3.7.2.1 Non-air-gapped (all environments)

In a non-air-gapped environment, your environment has two-way access to and from the **Internet**. Below are the prerequisites required if installing in a non-air-gapped environment:

- The version of the CLI that matches the DKP version you want to install.
- Review the Management Cluster Application Requirements (see page 122) and Workspace Platform Application Defaults and Resource Requirements (see page 124) to ensure that your cluster will have sufficient resources.
- Ensure you have a default StorageClass (see page 1531) configured (the Konvoy component is responsible for configuring one)
- A load balancer to route external traffic. In cloud environments, this is provided by your cloud provider. In on-premises and vSphere deployments, you can configure MetalLB. It is possible to use Virtual IP (see page 1100) also. See Load Balancing (see page 994) topic for more details.

NOTE: If you want to customize your cluster's domain or certificate, ensure you review the respective documentation sections:

- Configure custom domains and certificates during the Kommander installation (see page 1565)
- Configure custom domains and certificates after Kommander has been installed (see page 891)

- Ensure your firewall allows connections to github.com[184].
- For pre-provisioned on-premise environments:
    - Ensure you meet the Storage requirements (see page 108), default storage class (see page 110)and Application Storage (see page 1034).

184 http://github.com

- Ensure you have added at least 40 GB of raw storage to each of your worker nodes (see page 1037) in your clusters.

3.7.2.2 Air-gapped Only (**additional** prerequisites)

In an air-gapped environment, your environment is **isolated** from unsecured networks, like the Internet, and therefore require additional considerations for installation. Below are the **additional** prerequisites required if installing in an air-gapped environment:

- A local registry (see page 1606) containing all the necessary installation images, including the Kommander images which were downloaded in the air-gapped bundle above - `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`. See the pages for Load the Registry under each provider in the Day 1 - Basic Installs by Infrastructure (see page 146) section for how to push the necessary images to this registry.
- Connectivity with the clusters attaching to the management cluster:
  - Both management and attached clusters must be able to connect to the registry.
  - The management cluster must be able to connect to all attached cluster's API servers.
  - The management cluster must be able to connect to any load balancers created for platform services on the management cluster.

NOTE: If you want to customize your cluster's domain or certificate, ensure you review the respective documentation sections:

- Configure custom domains and certificates during the Kommander installation (see page 1565)
- Configure custom domains and certificates after Kommander has been installed (see page 891)

- For pre-provisioned on-premise environments:
  - Ensure you meet the Storage requirements[185].
  - Ensure you have added at least 40 GB of raw storage to each of your worker nodes (see page 1037) in your cluster.

3.7.2.3 Next Steps:

If using the Day 1- Basic Install instructions, proceed to that section Day 1 - Basic Installs by Infrastructure (see page 146) combinations to install and setup DKP based on your infrastructure environment provider.

185 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/213027274/2.5+Rook+Ceph+Configuration#Resource-Requirements-[inlineExtension]

If using the Custom Install instructions, proceed to that section and select the infrastructure provider you are using under Custom Installation and Additional Infrastructure Tools (see page 1050).

3.7.3 Container Runtimes

DKP supports both Podman and Docker as container engines for cluster creation. If you choose to use Podman, it works with Linux on DKP.

> Installation of Podman will be different for different OS environments, so please refer to the Podman Installation[186] site to install.

Version requirements for a container engine:

- Version 4.0 of Podman[187]or higher for Linux. Host requirements found here: https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
- Version 20.10.0 of Docker[188] or higher for Linux or MacOS. On macOS, Docker runs in a virtual machine. Configure this VM with at least 8 GB of memory.

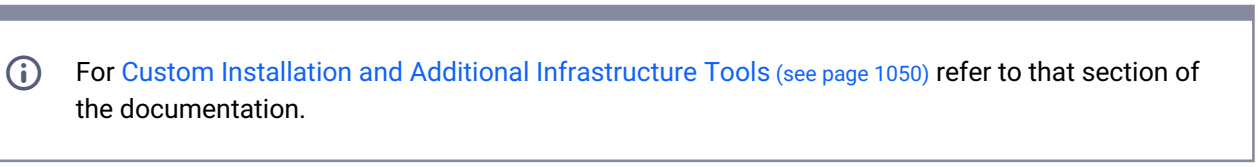
3.7.3.1 TIPS

- Set[189] `alias docker=podman` if desired to ensure any Docker specific commands will run without problem, but our testing shows it is not required.
- You will need to restart your machine after following the Host Requirements[190] to add the `/etc/systemd/system/user@.service.d/delegate.conf` and the `/etc/modules-load.d/iptables.conf` files.

186 https://podman.io/getting-started/installation
187 https://podman.io/getting-started/installation
188 https://www.docker.com/
189 https://docs.podman.io/en/latest/
190 https://kind.sigs.k8s.io/docs/user/rootless/

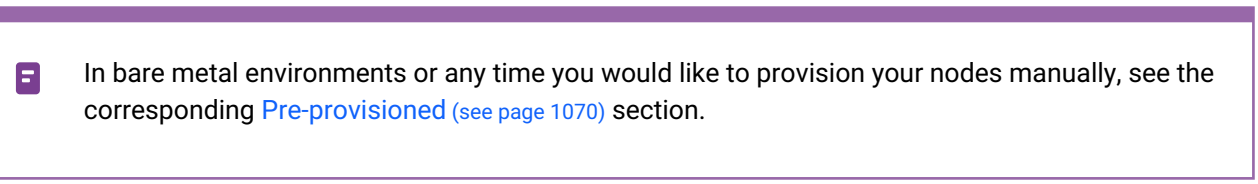
4 Day 1 - Basic Installs by Infrastructure

This section provides basic installation instructions for your infrastructure using combinations of providers and other variables. A **basic** cluster contains nodes and a running instance of DKP, but is not yet a *production* cluster. While you will not yet be ready to deploy a workload after these procedures, you will be able to get familiar with DKP and see the cluster structure. You can think of the install procedures in this section as a sort of quick start guide.



Depending on your selected infrastructure, and the requirements of your specific environment, there are additional configuration tasks in the Day 2 section (see page 590) to complete. Production cluster configuration lets you deploy and enable the Day 2 applications, and your workload applications, that you need for production operations.

For virtualized environments, DKP can provision the virtual machines necessary to run Kubernetes clusters. If you allow DKP to manage your infrastructure, look for your supported infrastructure provider installation choices below.



If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

4.1 Scenario Based Installation Instructions

4.2 Pre-provisioned Install Options

- Pre-provisioned Non-air-gapped Install (see page 152)
- Pre-provisioned Air-gapped Install (see page 169)
- Pre-provisioned FIPS Install (see page 194)
- Pre-provisioned FIPS Air-gapped Install (see page 214)
- Pre-provisioned with GPU Install (see page 239)

- Pre-provisioned Air-gapped with GPU Install (see page 262)

4.3 AWS Install Options

- AWS Install (see page 295)
- AWS Air-gapped Install (see page 312)
- AWS with FIPS Install (see page 333)
- AWS Air-gapped FIPS Install (see page 350)
- AWS with GPU Install (see page 369)
- AWS Air-gapped with GPU Install (see page 389)

4.4 EKS Install Options

- EKS Install (see page 413)

4.5 vSphere Install Options

- vSphere Prerequisites: All Installation Types (see page 442)
- vSphere Install (see page 447)
- vSphere Air-gapped Install (see page 468)
- vSphere FIPS Install (see page 492)
- vSphere FIPS Air-gapped Install (see page 514)

4.6 VMware Cloud Director Install Options

- Cloud Director Service Provider (see page 539)

4.7 Azure Install Options

- Azure Install (see page 541)

4.8 AKS Install Options

- AKS Install (see page 556)

4.9 GCP Install Options

- GCP Install (see page 572)

4.10 Pre-provisioned Install Options

For an either non-air-gapped or air-gapped environment that is Pre-provisioned (see page 84), install options are provided for you in this one location.

Remember, there are always more options for custom YAML in the Pre-provisioned Infrastructure (see page 1070) section, but this will get you operative with the most common scenarios.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

Below are all the DKP supported environment combinations: (Refer to this page to check if your OS is a Supported Operating System (see page 67).)

- Pre-provisioned Non-air-gapped Install (see page 152)
- Pre-provisioned Air-gapped Install (see page 169)
- Pre-provisioned FIPS Install (see page 194)
- Pre-provisioned FIPS Air-gapped Install (see page 214)
- Pre-provisioned with GPU Install (see page 239)
- Pre-provisioned Air-gapped with GPU Install (see page 262)

Important Pre-provisioned Topics:

- Pre-provisioned includes on-premises, vSphere, AWS, Azure, and GCP infrastructures and is described in more detail under What is Pre-provisioned Infrastructure (see page 84).
- For more information regarding CSI Disk storage and changing default StorageClass, see Default Storage Providers in DKP (see page 110).
- For Azure environment using Pre-provisioned specifics, see Pre-provisioned Azure only Configurations (see page 1079)

4.10.1 Resources

Additional resource information specific to On Premises and Pre-provisioned is listed below:

- **Control plane machines** -
  - 15% free space on the root file system.
  - Multiple ports open, as described in DKP Ports (see page 65).
  - `firewalld` systemd service disabled. If it exists and is enabled, use the commands `systemctl stop firewalld` then `systemctl disable firewalld`, so that `firewalld` remains disabled after the machine restarts.

Swap is disabled. The `kubelet` does not have generally-available support for swap. Due to variable commands, refer to your operating system documentation.

- **Worker machines** -
  - 15% free space on the root file system
  - If you plan to use local volume provisioning to provide persistent volumes for your workloads, you must mount at least four volumes to the `/mnt/disks/` mount point on each machine. Each volume must have at least 100 GiB of capacity.
  - Ensure your disk meets the resource requirements for Rook Ceph in `Block` mode for ObjectStorageDaemons[191] as specified in the requirements table (see page 1037).
  - Multiple ports open, as described in DKP Ports (see page 65).
  - `firewalld` systemd service disabled. If it exists and is enabled, use the commands `systemctl stop firewalld` then `systemctl disable firewalld`, so that `firewalld` remains disabled after the machine restarts.

Swap is disabled. The `kubelet` does not have generally-available support for swap. Due to variable commands, refer to your operating system documentation.

4.10.2 Pre-provisioned Non-air-gapped Install

This section provides instructions to install DKP in a Pre-provisioned non-air-gapped environment.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

4.10.2.1 Next Step:

Pre-provisioned: Define Infrastructure (see page 149)

4.10.2.2 Pre-provisioned: Define Infrastructure

Define the cluster hosts and infrastructure

191 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

Konvoy needs to know how to access your cluster hosts. This is done using inventory resources. For initial cluster creation, you must define a control-plane and at least one worker pool.

4.10.2.2.1 Define your Infrastructure

1. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_SECRET_NAME="$CLUSTER_NAME-ssh-key"
```

2. Use the following template to help you define your infrastructure. The environment variables that you set in the previous step automatically replace the variable names when the inventory YAML file is created.

```
cat <<EOF > preprovisioned_inventory.yaml
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-control-plane
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    # Create as many of these as needed to match your infrastructure
    # Note that the command line parameter --control-plane-replicas determines
how many control plane nodes will actually be used.
    #
    - address: $CONTROL_PLANE_1_ADDRESS
    - address: $CONTROL_PLANE_2_ADDRESS
    - address: $CONTROL_PLANE_3_ADDRESS
  sshConfig:
    port: 22
    # This is the username used to connect to your infrastructure. This user
must be root or
    # have the ability to use sudo without a password
    user: $SSH_USER
    privateKeyRef:
```

```
      # This is the name of the secret you created in the previous step. It
must exist in the same
      # namespace as this inventory object.
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-md-0
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    - address: $WORKER_1_ADDRESS
    - address: $WORKER_2_ADDRESS
    - address: $WORKER_3_ADDRESS
    - address: $WORKER_4_ADDRESS
  sshConfig:
    port: 22
    user: $SSH_USER
    privateKeyRef:
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
EOF
```

4.10.2.2.2 Next Step:

Pre-provisioned: Define Control Plane Endpoint (see page 151)

4.10.2.3 Pre-provisioned: Define Control Plane Endpoint

4.10.2.3.1 Define the Control Plane Endpoint for your cluster as well as connection mechanism

A control plane should have three, five, or seven nodes, so it can remain available if one or more nodes fail. A control plane with one node should not be used in production.

In addition, the control plane should have an endpoint that remains available if some nodes fail.

```
                        -------- cp1.example.com:6443
                        |
lb.example.com:6443 ----------- cp2.example.com:6443
                        |
                        -------- cp3.example.com:6443
```

In this example, the control plane endpoint host is `lb.example.com`, and the control plane endpoint port is `6443`. The control plane nodes are `cp1.example.com`, `cp2.example.com`, and `cp3.example.com`. The port of each API server is `6443`.

4.10.2.3.2 Select your Connection Mechanism

A virtual IP is the address that the client uses to connect to the service. A load balancer is the device that distributes the client connections to the backend servers. Before you create a new DKP cluster, choose an external load balancer(LB) or virtual IP.

- **External load balancer**

It is recommended that an external load balancer be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests.

- **Built-in virtual IP**

If an external load balancer is not available, use the built-in virtual IP (see page 1100). The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine.

4.10.2.3.3 Single-Node Control Plane

Do not use a single-node control plane in a production cluster.

A control plane with one node can use its single node as the endpoint, so you will not require an external load balancer, or a built-in virtual IP. At least one control plane node must always be running. Therefore, to upgrade a cluster with one control plane node, a spare machine must be available in the control plane inventory. This machine is used to provision the new node before the old node is deleted.

Modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1613).

When the API server endpoints are defined, you can create the cluster using the link in Next Step below.

4.10.2.3.4 Known Limitations

Be aware of these limitations in the current release of DKP.

The control plane endpoint port is also used as the API server port on each control plane machine. The default port is 6443. Before you create the cluster, ensure the port is available for use on each control plane machine.

4.10.2.3.5 Next Step:

Pre-provisioned: Create a New Cluster (see page 153)

4.10.2.4 Pre-provisioned: Create a Management Cluster

Create a new Pre-provisioned Kubernetes cluster in a non-air-gapped environment with the steps below.

4.10.2.4.1 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[192] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=preprovisioned-example
```

192 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

4.10.2.4.2 Create a Kubernetes Cluster

After you have defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster. This process creates a self-managed cluster to be used as the Management cluster.

> Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1100) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

> DKP uses local static provisioner as the default storage provider (see page 110) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[193] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

For Pre-provisioned environments, you define a set of nodes that already exist. During the cluster creation process, Konvoy Image Builder (see page 1626)(KIB) is built into DKP and automatically runs the machine configuration process (which KIB uses to build images for other providers) against the set of nodes that you defined. This results in your pre-existing or **pre-provisioned** nodes being configured properly.

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 149) previously created.

The create cluster command below includes the `--self-managed` flag. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

1. This command uses the default external load balancer (LB) option (see alternative Step 1 for virtual IP):

```
dkp create cluster preprovisioned \
  --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
```

193 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

```
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --self-managed
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

1. **ALTERNATIVE Virtual IP** - if you don't have an external LB, and want to use a VIRTUAL IP (see page 1100) provided by kube-vip, specify these flags example below:

   ```
   dkp create cluster preprovisioned \
       --cluster-name ${CLUSTER_NAME} \
       --control-plane-endpoint-host 196.168.1.10 \
       --virtual-ip-interface eth1
   ```

   The output from this command is shortened here for reading clarity, but should start like this:

   ```
   Generating cluster resources
   cluster.cluster.x-k8s.io/preprovisioned-example created
   cont.........
   ```

2. Use the wait command to monitor the cluster control-plane readiness:

   ```
   kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
   timeout=30m
   ```

   Output:

   ```
   cluster.cluster.x-k8s.io/preprovisioned-example condition met
   ```

Depending on the cluster size, it will take a few minutes to create.

When the command completes, you will have a running Kubernetes cluster! For bootstrap and custom YAML cluster creation, refer to the Additional Infrastructure Customization section of the documentation for Pre-provisioned: Pre-provisioned Infrastructure (see page 1070)

Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

If changing the Calico encapsulation (see page 1095), D2iQ recommends changing it after cluster creation, but before production.

If you need to increase Docker Hub's rate limit[194], use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

4.10.2.4.3 Audit logs

To modify Control Plane Audit log settings, see Configuring the Control Plane (see page 1613).

4.10.2.4.4 Further Steps:

For more customized cluster creation, see Custom Installation and Additional Infrastructure Tools (see page 1070). That section is for Pre-Provisioned Override Files (see page 1680), custom flags, and more that specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied.

4.10.2.4.5 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.10.2.4.6 Next Step:

Pre-provisioned: Configure MetalLB (see page 157)

194 https://docs.docker.com/docker-hub/download-rate-limit/

4.10.2.5 Pre-provisioned: Configure MetalLB

4.10.2.5.1 Create a MetalLB configmap for your pre-provisioned infrastructure.

Choose one of the following two protocols you want to use to announce service IPs. If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can continue the installation process with Pre-provisioned: Install Kommander (see page 159) .

To use MetalLB, create a MetalLB configMap for your Pre-provisioned infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

4.10.2.5.2 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

- MetalLB IP address ranges/CIDRs should be within the node's primary network subnet (see page 1065).
- MetalLB IP address ranges/CIDRs and node subnet should not conflict with the Kubernetes cluster pod and service subnets.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
```

```
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

After this completes, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.10.2.5.3 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need 4 pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

After this completes, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.10.2.5.3.1 Next Step:

Pre-provisioned: Install Kommander (see page 159)

4.10.2.6 Pre-provisioned: Install Kommander

4.10.2.6.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

4.10.2.6.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init > kommander.yaml
   ```

4. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP's default configuration ships Ceph with PVC based storage[195] which requires your CSI provider to support PVC

195 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/

with type `volumeMode: Block`. As this is not possible with the default local static provisioner (see page 110), you can install Ceph in host storage[196] mode.

You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

```
rook-ceph-cluster:
    enabled: true
    values: |
      cephClusterSpec:
        storage:
          storageClassDeviceSets: []
          useAllDevices: true
          useAllNodes: true
          deviceFilter: "<<value>>"
```

b. To assign specific storage devices on all nodes to the Ceph cluster:

```
rook-ceph-cluster:
    enabled: true
    values: |
      cephClusterSpec:
        storage:
          storageClassDeviceSets: []
          useAllNodes: true
          useAllDevices: false
          deviceFilter: "^sdb."
```

Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[197]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[198].

5. *If required:* Customize your `kommander.yaml`.

See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, etc.

196 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/
197 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
198 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

4.10.2.6.1.2 Enable DKP Catalog Applications and Install Kommander

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.7.0
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).

- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.10.2.6.1.3 Next Step:

Pre-provisioned: Verify Install and Log in to UI (see page 165)

4.10.2.7 Pre-provisioned: Verify Install and Log in to UI

Verify Kommander Install and Log in to the Dashboard UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.10.2.7.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

4.10.2.7.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

4.10.2.7.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

4.10.2.7.4 Next Step:

Pre-provisioned: Subsequent Cluster Creation (see page 167)

4.10.2.8 Pre-provisioned: Create Managed Clusters Using the DKP CLI

Enterprise Gov Advanced

After initial cluster creation, you have the ability to create additional clusters from the CLI. In a previous step, the new cluster was created as Self-managed which allows it to be a Management cluster or a stand alone

cluster. Subsequent new clusters are not self-managed as they will likely be Managed or Attached clusters to this Management Cluster.

When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

4.10.2.8.1 Make New Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.10.2.8.2 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[199] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

199 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
export CLUSTER_NAME=preprovisioned-additional
```

4.10.2.8.3 Create a Kubernetes Cluster

After you have defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster. This process creates a self-managed cluster to be used as the Management cluster.

Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1100) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

DKP uses local static provisioner as the default storage provider (see page 110) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[200] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

For Pre-provisioned environments, you define a set of nodes that already exist. During the cluster creation process, Konvoy Image Builder (see page 1626)(KIB) is built into DKP and automatically runs the machine configuration process (which KIB uses to build images for other providers) against the set of nodes that you defined. This results in your pre-existing or **pre-provisioned** nodes being configured properly.

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 149) previously created.

1. This command uses the default external load balancer (LB) option:

```
dkp create cluster preprovisioned
  --cluster-name ${CLUSTER_NAME}
  --control-plane-endpoint-host <control plane endpoint host>
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443>
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml
```

200 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

```
  --ssh-private-key-file <path-to-ssh-private-key>
```

2. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-additional condition met
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

NOTE: Depending on the cluster size, it will take a few minutes to create.

4.10.2.8.4 Manually Attach a DKP CLI Cluster to the Management Cluster

These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

```
echo ${MANAGED_CLUSTER_NAME}
```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
{MANAGED_CLUSTER_NAME}.conf
```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
   NOTE: This is only necessary if you never set the workspace of your cluster upon creation.
4. Retrieve the workspace where you want to attach the cluster:

```
kubectl get workspaces -A
```

5. Set the WORKSPACE_NAMESPACE environment variable:

```
export WORKSPACE_NAMESPACE=<workspace-namespace>
```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

```
kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
template='{{.data.value}}{{ "\n"}}'
```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
```

```
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

4.10.2.8.5 Next Step:

Day 2 - Cluster Operations Management (see page 590)

4.10.3 Pre-provisioned Air-gapped Install

This section provides instructions to install DKP in a Pre-provisioned air-gapped environment.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

> For air-gapped, ensure you have downloaded (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`, so you can extract the tarball on the page with those instructions.

4.10.3.1 Next Step:

Pre-provisioned Air-gapped: Configure Environment (see page 170)

4.10.3.2 Pre-provisioned Air-gapped: Configure Environment

The instructions below outline how to fulfill the requirements for using pre-provisioned infrastructure in an air-gapped environment. In order to create a cluster, you must first setup the environment with necessary artifacts. All artifacts for Pre-provisioned Air-gapped need to get onto the bastion host. Artifacts needed by nodes must be unpacked and distributed on the bastion before other provisioning will work in the absence of an internet connection.

There is an air-gapped bundle available to download (see page 76). In previous DKP releases, the distro package bundles were included in the downloaded air-gapped bundle. Currently, that air-gapped bundle contains the following artifacts with the exception of the distro packages:

- DKP Kubernetes packages
- Python packages (provided by upstream)
- Containerd tarball

1. Download (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz && cd dkp-v2.8.1/kib
   ```

2. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.
3. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle` . This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml` . Example command:

   ```
   ./konvoy-image create-package-bundle --os redhat-8.4 --output-
   directory=artifacts
   ```

   NOTE: For FIPS, pass the flag: `--fips`

   NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY` . Example command:

   ```
   export RHSM_ACTIVATION_KEY="-ci"
   export RHSM_ORG_ID="1232131"
   ```

4.10.3.2.1 Setup Process:

1. The bootstrap image must be extracted and loaded onto the bastion host (see page 1068).
2. Artifacts must be copied onto cluster hosts for nodes to access.

3. If using GPU, those artifacts must be positioned locally.
4. Registry seeded with images locally.

4.10.3.2.2 Load the Bootstrap Image

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` from the download site (see page 76) mentioned above and extracted the tarball, you will load the bootstrap.
2. Load the bootstrap image on your bastion machine:

```
docker load -i konvoy-bootstrap-image-v2.8.1.tar
```

4.10.3.2.3 Copy Air-gapped Artifacts onto Cluster Hosts

Using the Konvoy Image Builder (see page 1626), you can copy the required artifacts onto your cluster hosts.

Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , continue below:

1. The Kubernetes image bundle will be located in `kib/artifacts/images` and you will want to verify image and artifacts.
   a. Verify the image bundles exist in `artifacts/images` :

```
$ ls artifacts/images/
kubernetes-images-1.28.7-d2iq.1.tar kubernetes-images-1.28.7-d2iq.1-
fips.tar
```

   b. Verify the artifacts for your OS exist in the `artifacts/` directory and export the appropriate variables:

```
$ ls artifacts/
1.28.7_centos_7_x86_64.tar.gz       1.28.7_redhat_8_x86_64_fips.tar.gz
containerd-1.6.28-d2iq.1-rhel-7.9-x86_64.tar.gz   containerd-1.6.28-
d2iq.1-rhel-8.6-x86_64_fips.tar.gz  pip-packages.tar.gz
1.28.7_centos_7_x86_64_fips.tar.gz  1.28.7_rocky_9_x86_64.tar.gz
containerd-1.6.28-d2iq.1-rhel-7.9-x86_64_fips.tar.gz  containerd-1.6.28-
d2iq.1-rocky-9.0-x86_64.tar.gz
1.28.7_redhat_7_x86_64.tar.gz       1.28.7_ubuntu_20_x86_64.tar.gz
containerd-1.6.28-d2iq.1-rhel-8.4-x86_64.tar.gz   containerd-1.6.28-
d2iq.1-rocky-9.1-x86_64.tar.gz
1.28.7_redhat_7_x86_64_fips.tar.gz  containerd-1.6.28-d2iq.1-centos-7.9-
x86_64.tar.gz       containerd-1.6.28-d2iq.1-rhel-8.4-x86_64_fips.tar.gz
containerd-1.6.28-d2iq.1-ubuntu-20.04-x86_64.tar.gz
```

```
1.28.7_redhat_8_x86_64.tar.gz       containerd-1.6.28-d2iq.1-centos-7.9-
x86_64_fips.tar.gz  containerd-1.6.28-d2iq.1-rhel-8.6-x86_64.tar.gz
images
```

c. For example, for RHEL 8.4 you would set:

```
export OS_PACKAGES_BUNDLE=1.28.7_redhat_8_x86_64.tar.gz
export CONTAINERD_BUNDLE=containerd-1.6.28-d2iq.1-rhel-8.4-x86_64.tar.gz
```

2. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_FILE="<private key file>"
```

`SSH_PRIVATE_KEY_FILE` must be either the name of the SSH private key file in your working directory or an absolute path to the file in your user's home directory.

3. Generate an `inventory.yaml` which is automatically picked up by the `konvoy-image upload` in the next step.

```
cat <<EOF > inventory.yaml
all:
  vars:
    ansible_user: $SSH_USER
    ansible_port: 22
    ansible_ssh_private_key_file: $SSH_PRIVATE_KEY_FILE
  hosts:
    $CONTROL_PLANE_1_ADDRESS:
      ansible_host: $CONTROL_PLANE_1_ADDRESS
    $CONTROL_PLANE_2_ADDRESS:
      ansible_host: $CONTROL_PLANE_2_ADDRESS
    $CONTROL_PLANE_3_ADDRESS:
      ansible_host: $CONTROL_PLANE_3_ADDRESS
    $WORKER_1_ADDRESS:
      ansible_host: $WORKER_1_ADDRESS
    $WORKER_2_ADDRESS:
      ansible_host: $WORKER_2_ADDRESS
    $WORKER_3_ADDRESS:
      ansible_host: $WORKER_3_ADDRESS
    $WORKER_4_ADDRESS:
      ansible_host: $WORKER_4_ADDRESS
```

```
EOF
```

4. Upload the artifacts onto cluster hosts with the following command:

```
konvoy-image upload artifacts \
              --container-images-dir=./artifacts/images/ \
              --os-packages-bundle=./artifacts/$OS_PACKAGES_BUNDLE \
              --containerd-bundle=artifacts/$CONTAINERD_BUNDLE \
              --pip-packages-bundle=./artifacts/pip-packages.tar.gz
```

KIB uses variable overrides (see page 1670) to specify base image and container images to use in your new machine image. The variable overrides files for NVIDIA and FIPS can be ignored unless adding an overlay feature.

4.10.3.2.3.1 Next Step:

Pre-provisioned Air-gapped: Load the Registry (see page 173)

4.10.3.3 Pre-provisioned Air-gapped: Load the Registry

Before creating an air-gapped Kubernetes cluster, you need to load the required images in a local registry for the Konvoy component. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1606) page for more information.

1. If not already done in prerequisites, download (see page 76) the air-gapped bundle `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap cluster, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

```
cd dkp-v2.8.1
```

3. Set an environment variable with your registry address and any other needed variables using this command:

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
export REGISTRY_CA=<path to the cacert file on the bastion>
```

4. Execute the following command to load the air-gapped image bundle into your private registry using any of the relevant flags to apply variables above:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --
to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-
registry-password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

To increase Docker Hub's rate limit[201] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

4.10.3.3.1 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

4.10.3.3.2 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

201 https://docs.docker.com/docker-hub/download-rate-limit/

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

4.10.3.3.3 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME}
--to-registry-password=${REGISTRY_PASSWORD}
```

4.10.3.3.4 Next Step:

Pre-provisioned Air-gapped: Define Infrastructure (see page 175)

4.10.3.4 Pre-provisioned Air-gapped: Define Infrastructure

Define the cluster hosts and infrastructure

Konvoy needs to know how to access your cluster hosts. This is done using inventory resources. For initial cluster creation, you must define a control-plane and at least one worker pool.

4.10.3.4.1 Define your Infrastructure

1. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_SECRET_NAME="$CLUSTER_NAME-ssh-key"
```

2. Use the following template to help you define your infrastructure. The environment variables that you set in the previous step automatically replace the variable names when the inventory YAML file is created.

```
cat <<EOF > preprovisioned_inventory.yaml
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-control-plane
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    # Create as many of these as needed to match your infrastructure
    # Note that the command line parameter --control-plane-replicas determines
how many control plane nodes will actually be used.
    #
    - address: $CONTROL_PLANE_1_ADDRESS
    - address: $CONTROL_PLANE_2_ADDRESS
    - address: $CONTROL_PLANE_3_ADDRESS
  sshConfig:
    port: 22
    # This is the username used to connect to your infrastructure. This user
must be root or
    # have the ability to use sudo without a password
    user: $SSH_USER
    privateKeyRef:
      # This is the name of the secret you created in the previous step. It
must exist in the same
      # namespace as this inventory object.
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-md-0
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    - address: $WORKER_1_ADDRESS
    - address: $WORKER_2_ADDRESS
    - address: $WORKER_3_ADDRESS
    - address: $WORKER_4_ADDRESS
```

```
  sshConfig:
    port: 22
    user: $SSH_USER
    privateKeyRef:
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
EOF
```

4.10.3.4.2 Next Step:

Pre-provisioned Air-gapped: Define Control Plane Endpoint (see page 177)

4.10.3.5 Pre-provisioned Air-gapped: Define Control Plane Endpoint

4.10.3.5.1 Define the Control Plane Endpoint for your cluster as well as connection mechanism

A control plane should have three, five, or seven nodes, so it can remain available if one or more nodes fail. A control plane with one node should not be used in production.

In addition, the control plane should have an endpoint that remains available if some nodes fail.

```
                        -------- cp1.example.com:6443
                        |
lb.example.com:6443 ----------- cp2.example.com:6443
                        |
                        -------- cp3.example.com:6443
```

In this example, the control plane endpoint host is `lb.example.com`, and the control plane endpoint port is `6443`. The control plane nodes are `cp1.example.com`, `cp2.example.com`, and `cp3.example.com`. The port of each API server is `6443`.

4.10.3.5.2 Select your Connection Mechanism

A virtual IP is the address that the client uses to connect to the service. A load balancer is the device that distributes the client connections to the backend servers. Before you create a new DKP cluster, choose an external load balancer(LB) or virtual IP.

- **External load balancer**

It is recommended that an external load balancer be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests.

- **Built-in virtual IP**

If an external load balancer is not available, use the built-in virtual IP (see page 1100). The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine.

4.10.3.5.3 Single-Node Control Plane

Do not use a single-node control plane in a production cluster.

A control plane with one node can use its single node as the endpoint, so you will not require an external load balancer, or a built-in virtual IP. At least one control plane node must always be running. Therefore, to upgrade a cluster with one control plane node, a spare machine must be available in the control plane inventory. This machine is used to provision the new node before the old node is deleted.

Modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1613).

When the API server endpoints are defined, you can create the cluster using the link in Next Step below.

4.10.3.5.4 Known Limitations

Be aware of these limitations in the current release of DKP.

The control plane endpoint port is also used as the API server port on each control plane machine. The default port is 6443. Before you create the cluster, ensure the port is available for use on each control plane machine.

4.10.3.5.5 Next Step:

Pre-provisioned Air-gapped: Create a New Cluster (see page 178)

4.10.3.6 Pre-provisioned Air-gapped: Create a Management Cluster

If your cluster is air-gapped or you have a local registry (see page 1606), you must provide additional arguments when creating the cluster. These tell the cluster where to locate the local registry to use by defining the URL.

```
export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
export REGISTRY_CA=<path to the CA on the bastion>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

- `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.

4.10.3.6.1 Name Your Cluster

The cluster name may only contain the following characters: `a-z` , `0-9` , `.` , and `-` . Cluster creation will fail if the name has capital letters. See Kubernetes[202] for more naming information.

When specifying the `cluster-name` , you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-example>
```

Before you create a new DKP cluster below, choose an external load balancer(LB) or virtual IP (see page 1100) and use the corresponding `dkp create cluster` command.

202 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

4.10.3.6.2 Create an Air-gapped Kubernetes Cluster

After you have defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

> ⚠ DKP uses local static provisioner as the default storage provider (see page 110) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[203] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

> ⓘ For Pre-provisioned environments, you define a set of nodes that already exist. During the cluster creation process, Konvoy Image Builder (see page 1626)(KIB) is built into DKP and automatically runs the machine configuration process (which KIB uses to build images for other providers) against the set of nodes that you defined. This results in your pre-existing or **pre-provisioned** nodes being configured properly.

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory.

The create cluster command below includes the `--self-managed` flag. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

1. The command uses the default external load balancer (see alternative Step 1 below for virtual IP):

```
dkp create cluster preprovisioned --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
```

203 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

```
  --self-managed
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

1. **Virtual IP ALTERNATIVE** - if you don't have an external LB, and wish to use a VIRTUAL IP (see page 1100) provided by kube-vip, specify these flags example below:

   ```
   dkp create cluster preprovisioned \
       --cluster-name ${CLUSTER_NAME} \
       --control-plane-endpoint-host 196.168.1.10 \
       --virtual-ip-interface eth1
   ```

   The output from this command is shortened here for reading clarity, but should start like this:

   ```
   Generating cluster resources
   cluster.cluster.x-k8s.io/preprovisioned-example created
   cont.........
   ```

2. Use the wait command to monitor the cluster control-plane readiness:

   ```
   kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
   timeout=30m
   ```

   Output:

   ```
   cluster.cluster.x-k8s.io/preprovisioned-example condition met
   ```

NOTE: Depending on the cluster size, it will take a few minutes to create.

When the command completes, you will have a running Kubernetes cluster! For bootstrap and custom YAML cluster creation, refer to the Additional Infrastructure Customization section of the documentation for Pre-provisioned: Pre-provisioned Infrastructure (see page 1070)

Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

If changing the Calico encapsulation (see page 1095), D2iQ recommends changing it after cluster creation, but before production.

4.10.3.6.3 Audit Logs

To modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1613).

4.10.3.6.4 Further Steps:

For more customized cluster creation, see the Pre-Provisioned Additional Configurations (see page 1070) section.

NOTE: The section mentioned above is for overrides for your clusters (see page 1680), custom flags, and more that specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied.

4.10.3.6.5 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.10.3.6.6 Next Step:

Pre-provisioned Air-gapped: Configure MetalLB (see page 182)

4.10.3.7 Pre-provisioned Air-gapped: Configure MetalLB

4.10.3.7.1 Create a MetalLB configmap for your pre-provisioned infrastructure.

Choose one of the following two protocols you want to use to announce service IPs. If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can continue the installation process Pre-provisioned Air-gapped: Install Kommander (see page 184) .

To use MetalLB, create a MetalLB configMap for your Pre-provisioned infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

4.10.3.7.2 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.10.3.7.3 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need 4 pieces of information:

- The router IP address that MetalLB should connect to,

- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.10.3.7.3.1 Next Step:

Pre-provisioned Air-gapped: Install Kommander (see page 184)

4.10.3.8 Pre-provisioned Air-gapped: Install Kommander

4.10.3.8.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).

- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry: Air-gapped Environments (see page 1536).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

4.10.3.8.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init --airgapped > kommander.yaml
   ```

4. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP's default configuration ships Ceph with PVC based storage[204] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner (see page 110), you can install Ceph in host storage[205] mode.

   You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

   a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

      ```
      rook-ceph-cluster:
          enabled: true
          values: |
            cephClusterSpec:
              storage:
                storageClassDeviceSets: []
                useAllDevices: true
                useAllNodes: true
                deviceFilter: "<<value>>"
      ```

   b. To assign specific storage devices on all nodes to the Ceph cluster:

204 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
205 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/

```
rook-ceph-cluster:
    enabled: true
    values: |
      cephClusterSpec:
        storage:
          storageClassDeviceSets: []
          useAllNodes: true
          useAllDevices: false
          deviceFilter: "^sdb."
```

Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[206]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[207].

5. *If required:* Customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, etc.

4.10.3.8.1.2 Enable DKP Catalog Applications and Install Kommander in an Air-gapped Environment

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add the following values to enable DKP Catalog Applications:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./dkp-catalog-applications-v2.8.1.tar.gz
```

206 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
207 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.1.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.1.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.8.1.tar.gz
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.10.3.8.1.3 Next Step:

Pre-provisioned Air-gapped: Verify Install and Log in to UI (see page 187)

4.10.3.9 Pre-provisioned Air-gapped: Verify Install and Log in to UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.10.3.9.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

4.10.3.9.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

4.10.3.9.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as

attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

4.10.3.9.4 Next Step:

Pre-provisioned Air-gapped: Subsequent Cluster Creation (see page 190)

4.10.3.10 Pre-provisioned Air-gapped: Create Managed Clusters Using the DKP CLI

After initial cluster creation, you have the ability to create additional clusters from the CLI. In a previous step, the new cluster was created as Self-managed which allows it to be a Management cluster or a stand alone cluster. Subsequent new clusters are not self-managed as they will likely be Managed or Attached clusters to this Management Cluster.

When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

4.10.3.10.1 Make New Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.10.3.10.2 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[208] for more naming information.

208 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-additional>
```

4.10.3.10.3 Create a Kubernetes Cluster

After you have defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster. This process creates a self-managed cluster to be used as the Management cluster.

Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1100) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

DKP uses local static provisioner as the default storage provider (see page 110) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[209] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

For Pre-provisioned environments, you define a set of nodes that already exist. During the cluster creation process, Konvoy Image Builder (see page 1626)(KIB) is built into DKP and automatically runs the machine

209 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

configuration process (which KIB uses to build images for other providers) against the set of nodes that you defined. This results in your pre-existing or **pre-provisioned** nodes being configured properly.

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 149) previously created.

1. This command uses the default external load balancer (LB) option:

   ```
   dkp create cluster preprovisioned --cluster-name ${CLUSTER_NAME}
     --control-plane-endpoint-host <control plane endpoint host>
     --control-plane-endpoint-port <control plane endpoint port, if different than
   6443>
     --pre-provisioned-inventory-file preprovisioned_inventory.yaml
     --ssh-private-key-file <path-to-ssh-private-key>
     --registry-mirror-url=${REGISTRY_URL} \
     --registry-mirror-cacert=${REGISTRY_CA} \
     --registry-mirror-username=${REGISTRY_USERNAME} \
     --registry-mirror-password=${REGISTRY_PASSWORD}
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

2. Use the wait command to monitor the cluster control-plane readiness:

   ```
   kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
   timeout=30m
   ```

   Output:

   ```
   cluster.cluster.x-k8s.io/preprovisioned-additional condition met
   ```

NOTE: Depending on the cluster size, it will take a few minutes to create.

4.10.3.10.4 Manually Attach a DKP CLI Cluster to the Management Cluster

These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

```
echo ${MANAGED_CLUSTER_NAME}
```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
{MANAGED_CLUSTER_NAME}.conf
```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

```
kubectl get workspaces -A
```

5. Set the WORKSPACE_NAMESPACE environment variable:

```
export WORKSPACE_NAMESPACE=<workspace-namespace>
```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

```
kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
template='{{.data.value}}{{ "\n"}}'
```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
```

```
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

4.10.3.10.5 Next Step:

Day 2 - Cluster Operations Management (see page 590)

4.10.4 Pre-provisioned FIPS Install

This section provides instructions to install DKP in a Pre-provisioned non-air-gapped environment using FIPS.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

4.10.4.1 Next Step:

Pre-provisioned FIPS: Define Infrastructure (see page 195)

4.10.4.2 Pre-provisioned FIPS: Define Infrastructure

Define the cluster hosts and infrastructure

Konvoy needs to know how to access your cluster hosts. This is done using inventory resources. For initial cluster creation, you must define a control-plane and at least one worker pool.

4.10.4.2.1 Define your Infrastructure

1. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_SECRET_NAME="$CLUSTER_NAME-ssh-key"
```

2. Use the following template to help you define your infrastructure. The environment variables that you set in the previous step automatically replace the variable names when the inventory YAML file is created.

```
cat <<EOF > preprovisioned_inventory.yaml
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-control-plane
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    # Create as many of these as needed to match your infrastructure
    # Note that the command line parameter --control-plane-replicas determines
how many control plane nodes will actually be used.
    #
    - address: $CONTROL_PLANE_1_ADDRESS
```

```
    - address: $CONTROL_PLANE_2_ADDRESS
    - address: $CONTROL_PLANE_3_ADDRESS
  sshConfig:
    port: 22
    # This is the username used to connect to your infrastructure. This user
must be root or
    # have the ability to use sudo without a password
    user: $SSH_USER
    privateKeyRef:
      # This is the name of the secret you created in the previous step. It
must exist in the same
      # namespace as this inventory object.
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-md-0
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    - address: $WORKER_1_ADDRESS
    - address: $WORKER_2_ADDRESS
    - address: $WORKER_3_ADDRESS
    - address: $WORKER_4_ADDRESS
  sshConfig:
    port: 22
    user: $SSH_USER
    privateKeyRef:
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
EOF
```

4.10.4.2.2 Next Step:

Pre-provisioned FIPS: Define Control Plane Endpoint (see page 196)

4.10.4.3 Pre-provisioned FIPS: Define Control Plane Endpoint

4.10.4.3.1 Define the Control Plane Endpoint for your cluster as well as connection mechanism

A control plane should have three, five, or seven nodes, so it can remain available if one or more nodes fail. A control plane with one node should not be used in production.

In addition, the control plane should have an endpoint that remains available if some nodes fail.

```
                          -------- cp1.example.com:6443
                          |
      lb.example.com:6443 ---------- cp2.example.com:6443
                          |
                          -------- cp3.example.com:6443
```

In this example, the control plane endpoint host is `lb.example.com`, and the control plane endpoint port is `6443`. The control plane nodes are `cp1.example.com`, `cp2.example.com`, and `cp3.example.com`. The port of each API server is `6443`.

4.10.4.3.2 Select your Connection Mechanism

A virtual IP is the address that the client uses to connect to the service. A load balancer is the device that distributes the client connections to the backend servers. Before you create a new DKP cluster, choose an external load balancer(LB) or virtual IP.

- **External load balancer**

It is recommended that an external load balancer be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests.

- **Built-in virtual IP**

If an external load balancer is not available, use the built-in virtual IP (see page 1100). The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine.

4.10.4.3.3 Single-Node Control Plane

Do not use a single-node control plane in a production cluster.

A control plane with one node can use its single node as the endpoint, so you will not require an external load balancer, or a built-in virtual IP. At least one control plane node must always be running. Therefore, to upgrade a cluster with one control plane node, a spare machine must be available in the control plane inventory. This machine is used to provision the new node before the old node is deleted.

Modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1613).

When the API server endpoints are defined, you can create the cluster using the link in Next Step below.

4.10.4.3.4 Known Limitations

Be aware of these limitations in the current release of DKP.

The control plane endpoint port is also used as the API server port on each control plane machine. The default port is 6443. Before you create the cluster, ensure the port is available for use on each control plane machine.

4.10.4.3.5 Next Step:

Pre-provisioned FIPS: Create a New Cluster (see page 198)

4.10.4.4 Pre-provisioned FIPS: Create a Management Cluster

4.10.4.4.1 Deploying a Cluster in FIPS mode

In order to create a cluster in FIPS mode, we must inform the bootstrap controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes.

4.10.4.4.1.1 Supported FIPS Builds

Component	Repository	Version
Kubernetes	docker.io/mesosphere[210]	v1.28.7+fips.0
etcd	docker.io/mesosphere[211]	3.5.10+fips.0

210 https://hub.docker.com/layers/mesosphere/kube-apiserver/v1.28.7_d2iq.0/images/sha256-1cafe40f5a17c86aa75574b25dd637bbeb377e2ac703532d72f1118c1e966e28?context=explore

211 https://hub.docker.com/layers/mesosphere/etcd/v3.5.10_fips.0/images/sha256-60da8d0d3b37e71a4fa918ffa7ffd9963d9892b3ba43b8f63119d806f2e585ed?context=explore

4.10.4.4.1.2 Name Your Cluster

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[212] for more naming information.
>
> When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-example>
```

4.10.4.4.1.3 Create a Kubernetes Cluster

After you have defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster. This process creates a self-managed cluster to be used as the Management cluster.

> Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1100) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

212 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

DKP uses local static provisioner as the default storage provider (see page 110) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[213] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

For Pre-provisioned environments, you define a set of nodes that already exist. During the cluster creation process, Konvoy Image Builder (see page 1626)(KIB) is built into DKP and automatically runs the machine configuration process (which KIB uses to build images for other providers) against the set of nodes that you defined. This results in your pre-existing or **pre-provisioned** nodes being configured properly.

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 149) previously created.

The create cluster command below includes the `--self-managed` flag. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

1. This command uses the default external load balancer (LB) option (see alternative Step 1 below for virtual IP):

```
dkp create cluster preprovisioned \
  --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --kubernetes-version=v1.25.4+fips.0 \
  --etcd-version=3.5.6+fips.0 \
  --kubernetes-image-repository=docker.io/mesosphere \
  --self-managed
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

213 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

1. **ALTERNATIVE Virtual IP** - if you don't have an external LB, and want to use a VIRTUAL IP (see page 1100) provided by kube-vip, specify these flags example below:

   ```
   dkp create cluster preprovisioned \
       --cluster-name ${CLUSTER_NAME} \
       --control-plane-endpoint-host 196.168.1.10 \
       --virtual-ip-interface eth1
   ```

   The output from this command is shortened here for reading clarity, but should start like this:

   ```
   Generating cluster resources
   cluster.cluster.x-k8s.io/preprovisioned-example created
   cont.........
   ```

2. Use the wait command to monitor the cluster control-plane readiness:

   ```
   kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
   timeout=30m
   ```

   Output:

   ```
   cluster.cluster.x-k8s.io/preprovisioned-example condition met
   ```

Depending on the cluster size, it will take a few minutes to create.

When the command completes, you will have a running Kubernetes cluster! For bootstrap and custom YAML cluster creation, refer to the Additional Infrastructure Customization section of the documentation for Pre-provisioned: Pre-provisioned Infrastructure (see page 1070)

Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

If changing the Calico encapsulation (see page 1095), D2iQ recommends changing it after cluster creation, but before production.

If you need to increase Docker Hub's rate limit[214], use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

4.10.4.4.1.4 Audit logs

To modify Control Plane Audit log settings, see Configuring the Control Plane (see page 1613).

4.10.4.4.1.5 Further Steps:

For more customized cluster creation, see Custom Installation and Additional Infrastructure Tools (see page 1070). That section is for Pre-Provisioned Override Files (see page 1680), custom flags, and more that specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied.

For more information regarding FIPS, refer to the FIPS 140-2 Compliance (see page 1598) section of the documentation.

4.10.4.4.1.6 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.10.4.4.1.7 Next Step:

Pre-provisioned FIPS: Configure MetalLB (see page 202)

4.10.4.5 Pre-provisioned FIPS: Configure MetalLB

4.10.4.5.1 Create a MetalLB configmap for your pre-provisioned infrastructure.

Choose one of the following two protocols you want to use to announce service IPs. If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can continue the installation process.

To use MetalLB, create a MetalLB configMap for your Pre-provisioned infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

214 https://docs.docker.com/docker-hub/download-rate-limit/

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

4.10.4.5.2 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.10.4.5.3 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need 4 pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,

- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.10.4.5.3.1 Next Step:

Pre-provisioned FIPS: Install Kommander (see page 204)

4.10.4.6 Pre-provisioned FIPS: Install Kommander

4.10.4.6.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

4.10.4.6.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init > kommander.yaml
   ```

4. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP's default configuration ships Ceph with PVC based storage[215] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner (see page 110), you can install Ceph in host storage[216] mode.

   You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

   a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

      ```
      rook-ceph-cluster:
          enabled: true
          values: |
            cephClusterSpec:
              storage:
                storageClassDeviceSets: []
                useAllDevices: true
                useAllNodes: true
                deviceFilter: "<<value>>"
      ```

   b. To assign specific storage devices on all nodes to the Ceph cluster:

      ```
      rook-ceph-cluster:
          enabled: true
          values: |
            cephClusterSpec:
      ```

215 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
216 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/

```
        storage:
          storageClassDeviceSets: []
          useAllNodes: true
          useAllDevices: false
          deviceFilter: "^sdb."
```

Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[217]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[218].

5. *If required:* Customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, etc.

4.10.4.6.1.2 Enable DKP Catalog Applications and Install Kommander

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.7.0
```

⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

217 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
218 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.10.4.6.1.3 Next Step:

Pre-provisioned FIPS: Verify Install and Log in to UI (see page 207)

4.10.4.7 Pre-provisioned FIPS: Verify Install and Log in to UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.10.4.7.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

4.10.4.7.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

4.10.4.7.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

4.10.4.7.4 Next Step:

Pre-provisioned FIPS: Subsequent Cluster Creation (see page 210)

4.10.4.8 Pre-provisioned FIPS: Create Managed Clusters Using the DKP CLI

After initial cluster creation, you have the ability to create additional clusters from the CLI. In a previous step, the new cluster was created as Self-managed which allows it to be a Management cluster or a stand alone cluster. Subsequent new clusters are not self-managed as they will likely be Managed or Attached clusters to this Management Cluster.

When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

4.10.4.8.1 Make New Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.10.4.8.2 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[219] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

219 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-additional>
```

4.10.4.8.3 Create a Kubernetes Cluster

After you have defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster. This process creates a self-managed cluster to be used as the Management cluster.

Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1100) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

DKP uses local static provisioner as the default storage provider (see page 110) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[220] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

For Pre-provisioned environments, you define a set of nodes that already exist. During the cluster creation process, Konvoy Image Builder (see page 1626)(KIB) is built into DKP and automatically runs the machine configuration process (which KIB uses to build images for other providers) against the set of nodes that you defined. This results in your pre-existing or **pre-provisioned** nodes being configured properly.

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 149) previously created.

1. This command uses the default external load balancer (LB) option:

220 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

```
dkp create cluster preprovisioned \
  --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --kubernetes-version=v1.28.7+fips.0 \
  --etcd-version=3.5.10+fips.0 \
  --kubernetes-image-repository=docker.io/mesosphere
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

2. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-additional condition met
```

NOTE: Depending on the cluster size, it will take a few minutes to create.

4.10.4.8.4 Manually Attach a DKP CLI Cluster to the Management Cluster

These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

```
echo ${MANAGED_CLUSTER_NAME}
```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
{MANAGED_CLUSTER_NAME}.conf
```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

```
kubectl get workspaces -A
```

5. Set the WORKSPACE_NAMESPACE environment variable:

```
export WORKSPACE_NAMESPACE=<workspace-namespace>
```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

```
kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
template='{{.data.value}}{{ "\n"}}'
```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

4.10.4.8.5 Next Step:

Day 2 - Cluster Operations Management (see page 590)

4.10.5 Pre-provisioned FIPS Air-gapped Install

This section provides instructions to install DKP in a Pre-provisioned non-air-gapped environment using FIPS.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

For air-gapped, ensure you have downloaded (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , so you can extract the tarball on the page with those instructions.

4.10.5.1 Next Step:

Pre-provisioned Air-gapped FIPS: Configure Environment (see page 215)

4.10.5.2 Pre-provisioned Air-gapped FIPS: Configure Environment

In order to create a cluster in a Pre-provisioned Air-gapped environment with FIPS, you must first prepare the environment.

The instructions below outline how to fulfill the requirements for using pre-provisioned infrastructure in an air-gapped environment. In order to create a cluster, you must first setup the environment with necessary artifacts. All artifacts for Pre-provisioned Air-gapped need to get onto the bastion host. Artifacts needed by nodes must be unpacked and distributed on the bastion before other provisioning will work in the absence of an internet connection.

There is an air-gapped bundle available to download (see page 76). In previous DKP releases, the distro package bundles were included in the downloaded air-gapped bundle. Currently, that air-gapped bundle contains the following artifacts with the exception of the distro packages:

- DKP Kubernetes packages
- Python packages (provided by upstream)
- Containerd tarball

1. Download (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz && cd dkp-v2.8.1/kib
   ```

2. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.
3. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle` . This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml` . Example command:

   ```
   ./konvoy-image create-package-bundle --os redhat-8.4 --output-
   directory=artifacts
   ```

   NOTE: For FIPS, pass the flag: `--fips`

   NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY` . Example command:

   ```
   export RHSM_ACTIVATION_KEY="-ci"
   ```

```
export RHSM_ORG_ID="1232131"
```

4.10.5.2.1 Setup Process:

1. The bootstrap image must be extracted and loaded onto the bastion host (see page 1068).
2. Artifacts must be copied onto cluster hosts for nodes to access.
3. If using GPU, those artifacts must be positioned locally.
4. Registry seeded with images locally.

4.10.5.2.2 Load the Bootstrap Image

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` from the download site (see page 76) mentioned above and extracted the tarball, you will load the bootstrap.
2. Load the bootstrap image on your bastion machine:

```
docker load -i konvoy-bootstrap-image-v2.8.1.tar
```

4.10.5.2.3 Copy air-gapped artifacts onto cluster hosts

Using the Konvoy Image Builder (see page 1626), you can copy the required artifacts onto your cluster hosts.

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz` , extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz && cd dkp-v2.8.0/kib
```

2. The kubernetes image bundle will be located in `kib/artifacts/images` and you will want to verify image and artifacts.
   a. Verify the image bundles exist in `artifacts/images` :

```
$ ls artifacts/images/
kubernetes-images-1.28.7-d2iq.1.tar kubernetes-images-1.28.7-d2iq.1-
fips.tar
```

   b. Verify the artifacts for your OS exist in the `artifacts/` directory and export the appropriate variables:

```
$ ls artifacts/
1.28.7_centos_7_x86_64.tar.gz       1.28.7_redhat_8_x86_64_fips.tar.gz
containerd-1.6.28-d2iq.1-rhel-7.9-x86_64.tar.gz   containerd-1.6.28-
d2iq.1-rhel-8.6-x86_64_fips.tar.gz  pip-packages.tar.gz
1.28.7_centos_7_x86_64_fips.tar.gz  1.28.7_rocky_9_x86_64.tar.gz
containerd-1.6.28-d2iq.1-rhel-7.9-x86_64_fips.tar.gz  containerd-1.6.28-
d2iq.1-rocky-9.0-x86_64.tar.gz
1.28.7_redhat_7_x86_64.tar.gz       1.28.7_ubuntu_20_x86_64.tar.gz
containerd-1.6.28-d2iq.1-rhel-8.4-x86_64.tar.gz   containerd-1.6.28-
d2iq.1-rocky-9.1-x86_64.tar.gz
1.28.7_redhat_7_x86_64_fips.tar.gz  containerd-1.6.28-d2iq.1-centos-7.9-
x86_64.tar.gz       containerd-1.6.28-d2iq.1-rhel-8.4-x86_64_fips.tar.gz
containerd-1.6.28-d2iq.1-ubuntu-20.04-x86_64.tar.gz
1.28.7_redhat_8_x86_64.tar.gz       containerd-1.6.28-d2iq.1-centos-7.9-
x86_64_fips.tar.gz  containerd-1.6.28-d2iq.1-rhel-8.6-x86_64.tar.gz
images
```

c. For example, for RHEL 8.4 you would set:

```
export OS_PACKAGES_BUNDLE=1.28.7_redhat_8_x86_64_fips.tar.gz
export CONTAINERD_BUNDLE=containerd-1.6.10-d2iq.1-rhel-8.4-x86_64.tar.gz
```

3. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_FILE="<private key file>"
```

`SSH_PRIVATE_KEY_FILE` must be either the name of the SSH private key file in your working directory or an absolute path to the file in your user's home directory.

4. Generate an `inventory.yaml` which is automatically picked up by the `konvoy-image upload` in the next step.

```
cat <<EOF > inventory.yaml
all:
  vars:
    ansible_user: $SSH_USER
    ansible_port: 22
    ansible_ssh_private_key_file: $SSH_PRIVATE_KEY_FILE
```

```
  hosts:
    $CONTROL_PLANE_1_ADDRESS:
      ansible_host: $CONTROL_PLANE_1_ADDRESS
    $CONTROL_PLANE_2_ADDRESS:
      ansible_host: $CONTROL_PLANE_2_ADDRESS
    $CONTROL_PLANE_3_ADDRESS:
      ansible_host: $CONTROL_PLANE_3_ADDRESS
    $WORKER_1_ADDRESS:
      ansible_host: $WORKER_1_ADDRESS
    $WORKER_2_ADDRESS:
      ansible_host: $WORKER_2_ADDRESS
    $WORKER_3_ADDRESS:
      ansible_host: $WORKER_3_ADDRESS
    $WORKER_4_ADDRESS:
      ansible_host: $WORKER_4_ADDRESS
EOF
```

5. Upload the artifacts onto cluster hosts with the following command:

```
konvoy-image upload artifacts \
              --container-images-dir=./artifacts/images/ \
              --os-packages-bundle=./artifacts/$OS_PACKAGES_BUNDLE \
              --containerd-bundle=artifacts/$CONTAINERD_BUNDLE \
              --pip-packages-bundle=./artifacts/pip-packages.tar.gz
```

KIB uses variable overrides (see page 1670) to specify base image and container images to use in your new machine image. The variable overrides files for NVIDIA and FIPS can be ignored unless adding an overlay feature.

- Use the --overrides flag and reference either `fips.yaml` or `offline-fips.yaml` manifests located in the overrides directory[221] or see these pages in the documentation:
  - FIPS Overrides (see page 1671)
  - Create FIPS 140 Images (see page 1601)

4.10.5.2.3.1 Next Step:

Pre-provisioned Air-gapped FIPS: Load the Registry (see page 219)

221 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

4.10.5.3 Pre-provisioned Air-gapped FIPS: Load the Registry

Before creating an air-gapped Kubernetes cluster, you need to load the required images in a local registry for the Konvoy component. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1606) page for more information.

1. If not already done in prerequisites, download (see page 76) the air-gapped bundle `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap cluster, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

   ```
   cd dkp-v2.8.1
   ```

3. Set an environment variable with your registry address and any other needed variables using this command:

   ```
   export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   export REGISTRY_CA=<path to the cacert file on the bastion>
   ```

4. Execute the following command to load the air-gapped image bundle into your private registry using any of the relevant flags to apply variables above:

   ```
   dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --
   to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-
   registry-password=${REGISTRY_PASSWORD}
   ```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

4.10.5.3.1 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

4.10.5.3.2 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-password=${REGISTRY_PASSWORD}
```

4.10.5.3.3 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-password=${REGISTRY_PASSWORD}
```

To increase Docker Hub's rate limit[222] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

222 https://docs.docker.com/docker-hub/download-rate-limit/

4.10.5.3.4 Next Step:

Pre-provisioned Air-gapped FIPS: Define Infrastructure (see page 221)

4.10.5.4 Pre-provisioned Air-gapped FIPS: Define Infrastructure

Define the cluster hosts and infrastructure

Konvoy needs to know how to access your cluster hosts. This is done using inventory resources. For initial cluster creation, you must define a control-plane and at least one worker pool.

4.10.5.4.1 Define your Infrastructure

1. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_SECRET_NAME="$CLUSTER_NAME-ssh-key"
```

2. Use the following template to help you define your infrastructure. The environment variables that you set in the previous step automatically replace the variable names when the inventory YAML file is created.

```
cat <<EOF > preprovisioned_inventory.yaml
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-control-plane
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    # Create as many of these as needed to match your infrastructure
    # Note that the command line parameter --control-plane-replicas determines
how many control plane nodes will actually be used.
    #
    - address: $CONTROL_PLANE_1_ADDRESS
```

```
    - address: $CONTROL_PLANE_2_ADDRESS
    - address: $CONTROL_PLANE_3_ADDRESS
  sshConfig:
    port: 22
    # This is the username used to connect to your infrastructure. This user
must be root or
    # have the ability to use sudo without a password
    user: $SSH_USER
    privateKeyRef:
      # This is the name of the secret you created in the previous step. It
must exist in the same
      # namespace as this inventory object.
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-md-0
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    - address: $WORKER_1_ADDRESS
    - address: $WORKER_2_ADDRESS
    - address: $WORKER_3_ADDRESS
    - address: $WORKER_4_ADDRESS
  sshConfig:
    port: 22
    user: $SSH_USER
    privateKeyRef:
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
EOF
```

4.10.5.4.2 Next Step:

Pre-provisioned Air-gapped FIPS: Define Control Plane Endpoint (see page 222)

4.10.5.5 Pre-provisioned Air-gapped FIPS: Define Control Plane Endpoint

4.10.5.5.1 Define the Control Plane Endpoint for your cluster as well as connection mechanism

A control plane should have three, five, or seven nodes, so it can remain available if one or more nodes fail. A control plane with one node should not be used in production.

In addition, the control plane should have an endpoint that remains available if some nodes fail.

```
                          -------- cp1.example.com:6443
                          |
      lb.example.com:6443 ---------- cp2.example.com:6443
                          |
                          -------- cp3.example.com:6443
```

In this example, the control plane endpoint host is `lb.example.com`, and the control plane endpoint port is `6443`. The control plane nodes are `cp1.example.com`, `cp2.example.com`, and `cp3.example.com`. The port of each API server is `6443`.

4.10.5.5.2 Select your Connection Mechanism

A virtual IP is the address that the client uses to connect to the service. A load balancer is the device that distributes the client connections to the backend servers. Before you create a new DKP cluster, choose an external load balancer(LB) or virtual IP.

- **External load balancer**

It is recommended that an external load balancer be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests.

- **Built-in virtual IP**

If an external load balancer is not available, use the built-in virtual IP (see page 1100). The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine.

4.10.5.5.3 Single-Node Control Plane

Do not use a single-node control plane in a production cluster.

A control plane with one node can use its single node as the endpoint, so you will not require an external load balancer, or a built-in virtual IP. At least one control plane node must always be running. Therefore, to upgrade a cluster with one control plane node, a spare machine must be available in the control plane inventory. This machine is used to provision the new node before the old node is deleted.

Modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1613).

When the API server endpoints are defined, you can create the cluster using the link in Next Step below.

4.10.5.5.4 Known Limitations

Be aware of these limitations in the current release of DKP.

The control plane endpoint port is also used as the API server port on each control plane machine. The default port is 6443. Before you create the cluster, ensure the port is available for use on each control plane machine.

4.10.5.5.5 Next Step:

Pre-provisioned Air-gapped FIPS: Create a New Cluster (see page 224)

4.10.5.6 Pre-provisioned Air-gapped FIPS: Create a Management Cluster

If your cluster is air-gapped or you have a local registry (see page 1606), you must provide additional arguments when creating the cluster. These tell the cluster where to locate the local registry to use by defining the URL.

```
export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
export REGISTRY_CA=<path to the CA on the bastion>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

- `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.

4.10.5.6.1 Name Your Cluster

The cluster name may only contain the following characters: `a-z` , `0-9` , `.` , and `-` . Cluster creation will fail if the name has capital letters. See Kubernetes[223] for more naming information.

223 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-example>
```

4.10.5.6.2 Create an Air-gapped Kubernetes Cluster

After you have defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

DKP uses local static provisioner as the default storage provider (see page 110) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[224] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

For Pre-provisioned environments, you define a set of nodes that already exist. During the cluster creation process, Konvoy Image Builder (see page 1626)(KIB) is built into DKP and automatically runs the machine configuration process (which KIB uses to build images for other providers) against the set of nodes that you defined. This results in your pre-existing or **pre-provisioned** nodes being configured properly.

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory.

224 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

The create cluster command below includes the `--self-managed` flag. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1100) and use the corresponding `dkp create cluster` command.

1. The command below uses the default external load balancer (See alternative Step 1 for virtual IP):

```
dkp create cluster preprovisioned --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --kubernetes-version=v1.28.7+fips.0 \
  --etcd-version=3.5.10+fips.0 \
  --kubernetes-image-repository=docker.io/mesosphere \
  --self-managed
```


If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

1. **Virtual IP ALTERNATIVE** - if you don't have an external LB, and wish to use a VIRTUAL IP (see page 1100) provided by kube-vip, specify these flags example below:

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --control-plane-endpoint-host 196.168.1.10 \
    --virtual-ip-interface eth1
```

The output from this command is shortened here for reading clarity, but should start like this:

```
Generating cluster resources
```

```
cluster.cluster.x-k8s.io/preprovisioned-example created
cont.........
```

2. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-example condition met
```

NOTE: Depending on the cluster size, it will take a few minutes to create.

When the command completes, you will have a running Kubernetes cluster! For bootstrap and custom YAML cluster creation, refer to the Additional Infrastructure Customization section of the documentation for Pre-provisioned: Pre-provisioned Infrastructure (see page 1070)

Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

If changing the Calico encapsulation (see page 1095), D2iQ recommends changing it after cluster creation, but before production.

4.10.5.6.3 Audit Logs

To modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1613).

4.10.5.6.4 Further Steps:

For more customized cluster creation, see the Pre-Provisioned Additional Configurations (see page 1070) section.

NOTE: The section mentioned above is for overrides for your clusters (see page 1680), custom flags, and more that specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied.

4.10.5.6.5 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.10.5.6.6 Next Step:

Pre-provisioned Air-gapped FIPS: Configure MetalLB (see page 228)

4.10.5.7 Pre-provisioned Air-gapped FIPS: Configure MetalLB

4.10.5.7.1 Create a MetalLB configmap for your pre-provisioned infrastructure.

Choose one of the following two protocols you want to use to announce service IPs. If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can continue the installation process.

To use MetalLB, create a MetalLB configMap for your Pre-provisioned infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

4.10.5.7.2 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.10.5.7.3 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need 4 pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
```

```
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.10.5.7.3.1 Next Step:

Pre-provisioned Air-gapped FIPS: Install Kommander (see page 230)

4.10.5.8 Pre-provisioned Air-gapped FIPS: Install Kommander

4.10.5.8.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry: Air-gapped Environments (see page 1536).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

4.10.5.8.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

```
dkp install kommander --init --airgapped > kommander.yaml
```

4. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP's default configuration ships Ceph with PVC based storage[225] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner (see page 110), you can install Ceph in host storage[226] mode.

   You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

   a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

   ```
   rook-ceph-cluster:
       enabled: true
       values: |
         cephClusterSpec:
           storage:
             storageClassDeviceSets: []
             useAllDevices: true
             useAllNodes: true
             deviceFilter: "<<value>>"
   ```

   b. To assign specific storage devices on all nodes to the Ceph cluster:

   ```
   rook-ceph-cluster:
       enabled: true
       values: |
         cephClusterSpec:
           storage:
             storageClassDeviceSets: []
             useAllNodes: true
             useAllDevices: false
             deviceFilter: "^sdb."
   ```

   Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[227]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[228].

5. *If required:* Customize your `kommander.yaml`.

   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, etc.

225 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
226 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/
227 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
228 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

4.10.5.8.1.2 Enable DKP Catalog Applications and Install Kommander in an Air-gapped Environment

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add the following values to enable DKP Catalog Applications:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./dkp-catalog-applications-v2.8.1.tar.gz
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.1.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.1.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.8.1.tar.gz
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.10.5.8.1.3 Next Step:

Pre-provisioned Air-gapped FIPS: Verify Install and Log in to UI (see page 233)

4.10.5.9 Pre-provisioned Air-gapped FIPS: Verify Install and Log in to UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.10.5.9.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

4.10.5.9.2 Next Step:

Pre-provisioned Air-gapped FIPS: Subsequent Clusters (see page 234)

4.10.5.10 Pre-provisioned Air-gapped FIPS: Create Managed Clusters Using the DKP CLI

After initial cluster creation, you have the ability to create additional clusters from the CLI. In a previous step, the new cluster was created as Self-managed which allows it to be a Management cluster or a stand alone cluster. Subsequent new clusters are not self-managed as they will likely be Managed or Attached clusters to this Management Cluster.

When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

4.10.5.10.1 Make New Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.10.5.10.2 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[229] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-additional>
```

229 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

4.10.5.10.3 Create a Kubernetes Cluster

After you have defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster. This process creates a self-managed cluster to be used as the Management cluster.

> Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1100) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

> DKP uses local static provisioner as the default storage provider (see page 110) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[230] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

For Pre-provisioned environments, you define a set of nodes that already exist. During the cluster creation process, Konvoy Image Builder (see page 1626)(KIB) is built into DKP and automatically runs the machine configuration process (which KIB uses to build images for other providers) against the set of nodes that you defined. This results in your pre-existing or **pre-provisioned** nodes being configured properly.

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 149) previously created.

1. This command uses the default external load balancer (LB) option:

```
dkp create cluster preprovisioned --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
```

230 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

```
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD}
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

2. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-additional condition met
```

NOTE: Depending on the cluster size, it will take a few minutes to create.

4.10.5.10.4 Manually Attach a DKP CLI Cluster to the Management Cluster

These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

```
echo ${MANAGED_CLUSTER_NAME}
```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
{MANAGED_CLUSTER_NAME}.conf
```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
   NOTE: This is only necessary if you never set the workspace of your cluster upon creation.
4. Retrieve the workspace where you want to attach the cluster:

```
kubectl get workspaces -A
```

5. Set the WORKSPACE_NAMESPACE environment variable:

```
export WORKSPACE_NAMESPACE=<workspace-namespace>
```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

```
kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
template='{{.data.value}}{{ "\n"}}'
```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
```

```
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

4.10.5.10.5 Next Step:

Day 2 - Cluster Operations Management (see page 590)

4.10.6 Pre-provisioned with GPU Install

This section provides instructions to install DKP in a Pre-provisioned non-air-gapped environment using GPU.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

4.10.6.1 Next Step:

Pre-provisioned GPU: Nodepool Secrets and Overrides (see page 239)

4.10.6.2 Pre-provisioned GPU: Nodepool Secrets and Overrides

Install NVIDIA runfile and place it in the artifacts directory

4.10.6.2.1 For pre-provisioned environments, DKP has introduced the `nvidia-runfile` flag for Air-gapped Pre-provisioned environments. If the NVIDIA runfile[231] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory (you must create an `artifacts` directory if it doesn't already exist).

```
curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/NVIDIA-Linux-x86_64-470.82.01.run
mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
```

DKP supported NVIDIA driver[232] version is 470.x.

1. Create the secret that GPU nodepool would use, this secret is populated from the KIB overrides. In this example we have a file called, `overrides/nvidia.yaml`. It should resemble this:

```
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
```

2. Create a secret on the bootstrap cluster that is populated from the above file. We will name it

`${CLUSTER_NAME}-user-overrides`

```
kubectl create secret generic ${CLUSTER_NAME}-user-overrides --from-file=overrides.yaml=overrides/nvidia.yaml
```

3. Create an inventory and nodepool with the instructions below and use the `$CLUSTER_NAME-user-overrides` secret.

Follow these steps:

231 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile
232 https://www.nvidia.com/Download/Find.aspx

1. Create an inventory object that has the same name as the node pool you're creating, and the details of the pre-provisioned machines that you want to add to it. For example, to create a node pool named `gpu-nodepool` an inventory named `gpu-nodepool` must be present in the same namespace:

   ```
   apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
   kind: PreprovisionedInventory
   metadata:
     name: ${MY_NODEPOOL_NAME}
   spec:
     hosts:
       - address: ${IP_OF_NODE}
     sshConfig:
       port: 22
       user: ${SSH_USERNAME}
       privateKeyRef:
         name: ${NAME_OF_SSH_SECRET}
         namespace: ${NAMESPACE_OF_SSH_SECRET}
   ```

2. (Optional) If your pre-provisioned machines have overrides, you must create a secret that includes all of the overrides you want to provide in one file. Create an override secret using the instructions detailed on this page.

3. Once the `PreprovisionedInventory` object and overrides are created, create a node pool:

   ```
   dkp create nodepool preprovisioned -c ${MY_CLUSTER_NAME} $
   {MY_NODEPOOL_NAME} --override-secret-name ${MY_OVERRIDE_SECRET}
   ```

- Advanced users can use a combination of the `--dry-run` and `--output=yaml` or `--output-directory=<existing-directory>` flags to get a complete set of node pool objects to modify locally or store in version control.

For more information regarding this flag or others, please refer to the dkp create nodepool section of the documentation for either cluster or nodepool and select your provider.

For more information, see:

- Pre-provisioned Create and Delete Node Pools
- Pre-provisioned Air-gapped Define Environment

Next Step

Pre-provisioned GPU: Define Infrastructure (see page 242)

4.10.6.3 Pre-provisioned GPU: Define Infrastructure

Define the cluster hosts and infrastructure

Konvoy needs to know how to access your cluster hosts. This is done using inventory resources. For initial cluster creation, you must define a control-plane and at least one worker pool.

4.10.6.3.1 Define your Infrastructure

1. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_SECRET_NAME="$CLUSTER_NAME-ssh-key"
```

2. Use the following template to help you define your infrastructure. The environment variables that you set in the previous step automatically replace the variable names when the inventory YAML file is created.

```
cat <<EOF > preprovisioned_inventory.yaml
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-control-plane
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    # Create as many of these as needed to match your infrastructure
    # Note that the command line parameter --control-plane-replicas determines
how many control plane nodes will actually be used.
    #
    - address: $CONTROL_PLANE_1_ADDRESS
    - address: $CONTROL_PLANE_2_ADDRESS
    - address: $CONTROL_PLANE_3_ADDRESS
```

```
  sshConfig:
    port: 22
    # This is the username used to connect to your infrastructure. This user
must be root or
    # have the ability to use sudo without a password
    user: $SSH_USER
    privateKeyRef:
      # This is the name of the secret you created in the previous step. It
must exist in the same
      # namespace as this inventory object.
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-md-0
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    - address: $WORKER_1_ADDRESS
    - address: $WORKER_2_ADDRESS
    - address: $WORKER_3_ADDRESS
    - address: $WORKER_4_ADDRESS
  sshConfig:
    port: 22
    user: $SSH_USER
    privateKeyRef:
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
EOF
```

4.10.6.3.2 Next Step:

Pre-provisioned GPU: Define Control Plane Endpoint (see page 243)

4.10.6.4 Pre-provisioned GPU: Define Control Plane Endpoint

4.10.6.4.1 Define the Control Plane Endpoint for your cluster as well as connection mechanism

A control plane should have three, five, or seven nodes, so it can remain available if one or more nodes fail. A control plane with one node should not be used in production.

In addition, the control plane should have an endpoint that remains available if some nodes fail.

```
                          -------- cp1.example.com:6443
                          |
      lb.example.com:6443 ---------- cp2.example.com:6443
                          |
                          -------- cp3.example.com:6443
```

In this example, the control plane endpoint host is `lb.example.com`, and the control plane endpoint port is `6443`. The control plane nodes are `cp1.example.com`, `cp2.example.com`, and `cp3.example.com`. The port of each API server is `6443`.

4.10.6.4.2 Select your Connection Mechanism

A virtual IP is the address that the client uses to connect to the service. A load balancer is the device that distributes the client connections to the backend servers. Before you create a new DKP cluster, choose an external load balancer(LB) or virtual IP.

- **External load balancer**

It is recommended that an external load balancer be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests.

- **Built-in virtual IP**

If an external load balancer is not available, use the built-in virtual IP (see page 1100). The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine.

4.10.6.4.3 Single-Node Control Plane

Do not use a single-node control plane in a production cluster.

A control plane with one node can use its single node as the endpoint, so you will not require an external load balancer, or a built-in virtual IP. At least one control plane node must always be running. Therefore, to upgrade a cluster with one control plane node, a spare machine must be available in the control plane inventory. This machine is used to provision the new node before the old node is deleted.

Modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1613).

When the API server endpoints are defined, you can create the cluster using the link in Next Step below.

4.10.6.4.4 Known Limitations

Be aware of these limitations in the current release of DKP.

The control plane endpoint port is also used as the API server port on each control plane machine. The default port is 6443. Before you create the cluster, ensure the port is available for use on each control plane machine.

4.10.6.4.5 Next Step:

Pre-provisioned GPU: Create a Cluster (see page 245)

4.10.6.5 Pre-provisioned GPU: Create a Management Cluster

4.10.6.5.1 Create a Cluster with GPU AMI

If a custom AMI was created using Konvoy Image Builder, the custom `ami` id is printed and written to `packer.pkr.hcl`.

To use the built `ami` with Konvoy, specify it with the `--ami` flag when calling cluster create.

For GPU Steps in Pre-provisioned (see page 0) section of the documentation to use the overrides/nvidia.yaml.

Additional helpful information can be found in the NVIDIA Device Plug-in[233] for Kubernetes instructions and the Installation Guide of Supported Platforms[234].

4.10.6.5.2 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[235] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

233 https://github.com/NVIDIA/k8s-device-plugin/blob/master/README.md
234 https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html
235 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-example>
```

4.10.6.5.3 Create a Kubernetes Cluster

After you have defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster. This process creates a self-managed cluster to be used as the Management cluster.

Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1100) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

DKP uses local static provisioner as the default storage provider (see page 110) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[236] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

For Pre-provisioned environments, you define a set of nodes that already exist. During the cluster creation process, Konvoy Image Builder (see page 1626)(KIB) is built into DKP and automatically runs the machine configuration process (which KIB uses to build images for other providers) against the set of nodes that you defined. This results in your pre-existing or **pre-provisioned** nodes being configured properly.

236 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 149) previously created.

The create cluster command below includes the `--self-managed` flag. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

1. Execute this command to create a cluster with a GPU AMI using the default external load balancer option:

```
dkp create cluster preprovisioned \
  --cluster-name=${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --ami <ami> \
  --self-managed
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

1. **Virtual IP ALTERNATIVE** - if you don't have an external LB, and wish to use a VIRTUAL IP (see page 1100) provided by kube-vip, specify these flags example below:

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --control-plane-endpoint-host 196.168.1.10 \
    --virtual-ip-interface eth1
```

The output from this command is shortened here for reading clarity, but should start like this:

```
Generating cluster resources
cluster.cluster.x-k8s.io/preprovisioned-example created
cont.........
```

2. Create the node pool after cluster creation:

```
dkp create nodepool aws -c ${CLUSTER_NAME} \
```

```
--instance-type p2.xlarge \
--ami-id=${AMI_ID_FROM_KIB} \
--replicas=1 ${NODEPOOL_NAME} \
--kubeconfig=${CLUSTER_NAME}.conf
```

3. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-example condition met
```

Depending on the cluster size, it will take a few minutes to create.

When the command completes, you will have a running Kubernetes cluster! For bootstrap and custom YAML cluster creation, refer to the Additional Infrastructure Customization section of the documentation for Pre-provisioned: Pre-provisioned Infrastructure (see page 1070)

Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

If changing the Calico encapsulation (see page 1095), D2iQ recommends changing it after cluster creation, but before production.

To increase Docker Hub's rate limit[237] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

237 https://docs.docker.com/docker-hub/download-rate-limit/

4.10.6.5.4 Audit logs

To modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1613).

4.10.6.5.4.1 Further Steps:

For more customized cluster creation, access the Pre-Provisioned Additional Configurations (see page 1070) section. That section is for Pre-Provisioned Override Files (see page 245), custom flags, and more that specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied.

4.10.6.5.4.2 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.10.6.5.4.3 Next Step:

Pre-provisioned GPU: Configure MetalLB (see page 249)

4.10.6.6 Pre-provisioned GPU: Configure MetalLB

4.10.6.6.1 Create a MetalLB configmap for your pre-provisioned infrastructure.

Choose one of the following two protocols you want to use to announce service IPs. If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can continue the installation process.

To use MetalLB, create a MetalLB configMap for your Pre-provisioned infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

4.10.6.6.2 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.10.6.6.3 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need 4 pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.10.6.6.3.1 Next Step:

Pre-provisioned GPU: Install Kommander (see page 251)

4.10.6.7 Pre-provisioned GPU: Install Kommander

4.10.6.7.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

4.10.6.7.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1558) for the deployment:

```
dkp install kommander --init > kommander.yaml
```

4. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP's default configuration ships Ceph with PVC based storage[238] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner (see page 110), you can install Ceph in host storage[239] mode.

   You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

   a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

```
rook-ceph-cluster:
    enabled: true
    values: |
      cephClusterSpec:
        storage:
          storageClassDeviceSets: []
          useAllDevices: true
          useAllNodes: true
          deviceFilter: "<<value>>"
```

   b. To assign specific storage devices on all nodes to the Ceph cluster:

```
rook-ceph-cluster:
    enabled: true
    values: |
      cephClusterSpec:
        storage:
          storageClassDeviceSets: []
          useAllNodes: true
          useAllDevices: false
          deviceFilter: "^sdb."
```

   Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[240]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[241].

238 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
239 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/
240 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
241 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

5. *If required:* Customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, etc.

4.10.6.7.1.2 Enable GPU Resources

1. In the same `kommander.yaml` file, enable Nvidia platform services.

```
apps:
  nvidia-gpu-operator:
   enabled: true
```

2. Append the correct Toolkit version based on your OS:

   The **NVIDIA Container Toolkit** allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

   Centos 7.9/RHEL 7.9:
   If you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.14.6-centos7
```

   RHEL 8.4/8.6 and SLES 15 SP3
   If you're using RHEL 8.4/8.6 or SLES 15 SP3 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.14.6-ubi8
```

Ubuntu 18.04 and 20.04
If you're using Ubuntu 18.04 or 20.04 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.14.6-ubuntu20.04
```

4.10.6.7.1.3 Enable DKP Catalog Applications and Install Kommander

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.7.0
```

⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.10.6.7.1.4 Next Step:

Pre-provisioned GPU: Verify Install and Log in to UI (see page 255)

4.10.6.8 Pre-provisioned GPU: Verify Install and Log in to UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.10.6.8.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

4.10.6.8.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

4.10.6.8.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

For more GPU information (see page 998), see the section for GPU under Day 2 - Cluster Operations Management.

4.10.6.8.4 Next Step:

Pre-provisioned GPU: Subsequent Cluster Creation (see page 258)

4.10.6.9 Pre-provisioned GPU: Create Managed Clusters Using the DKP CLI

After initial cluster creation, you have the ability to create additional clusters from the CLI. In a previous step, the new cluster was created as Self-managed which allows it to be a Management cluster or a stand alone cluster. Subsequent new clusters are not self-managed as they will likely be Managed or Attached clusters to this Management Cluster.

When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

4.10.6.9.1 Make New Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.10.6.9.2 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[242] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

242 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-additional>
```

If a custom AMI was created using Konvoy Image Builder, the custom `ami` id is printed and written to `packer.pkr.hcl`. To use the built `ami` with Konvoy, specify it with the `--ami` flag when calling cluster create command below.

4.10.6.9.3 Create a Kubernetes Cluster

After you have defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster. This process creates a self-managed cluster to be used as the Management cluster.

Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1100) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

DKP uses local static provisioner as the default storage provider (see page 110) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[243] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

For Pre-provisioned environments, you define a set of nodes that already exist. During the cluster creation process, Konvoy Image Builder (see page 1626)(KIB) is built into DKP and automatically runs the machine configuration process (which KIB uses to build images for other providers) against the set of nodes that you defined. This results in your pre-existing or **pre-provisioned** nodes being configured properly.

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 149) previously created.

243 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

1. This command uses the default external load balancer (LB) option:

```
dkp create cluster preprovisioned
  --cluster-name ${CLUSTER_NAME}
  --control-plane-endpoint-host <control plane endpoint host>
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443>
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml
  --ssh-private-key-file <path-to-ssh-private-key>
  --ami <ami>
```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

2. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

   Output:

```
cluster.cluster.x-k8s.io/preprovisioned-additional condition met
```

NOTE: Depending on the cluster size, it will take a few minutes to create.

4.10.6.9.4 Manually Attach a DKP CLI Cluster to the Management Cluster

These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

```
echo ${MANAGED_CLUSTER_NAME}
```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
{MANAGED_CLUSTER_NAME}.conf
```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

```
kubectl get workspaces -A
```

5. Set the WORKSPACE_NAMESPACE environment variable:

```
export WORKSPACE_NAMESPACE=<workspace-namespace>
```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

```
kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
template='{{.data.value}}{{ "\n"}}'
```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

4.10.6.9.5 Next Step:

Day 2 - Cluster Operations Management (see page 590)

4.10.7 Pre-provisioned Air-gapped with GPU Install

This section provides instructions to install DKP in a Pre-provisioned air-gapped environment using GPU.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

For air-gapped, ensure you have downloaded (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , so you can extract the tarball on the page with those instructions.

4.10.7.1 Next Step:

Pre-provisioned Air-gapped GPU: Configure Environment (see page 263)

4.10.7.2 Pre-provisioned Air-gapped GPU: Configure Environment

If the NVIDIA runfile[244] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory.

- ```
  curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/
  NVIDIA-Linux-x86_64-470.82.01.run
  mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
  ```

The instructions below outline how to fulfill the requirements for using pre-provisioned infrastructure in an air-gapped environment. In order to create a cluster, you must first setup the environment with necessary artifacts. All artifacts for Pre-provisioned Air-gapped need to get onto the bastion host. Artifacts needed by nodes must be unpacked and distributed on the bastion before other provisioning will work in the absence of an internet connection.

There is an air-gapped bundle available to download (see page 76). In previous DKP releases, the distro package bundles were included in the downloaded air-gapped bundle. Currently, that air-gapped bundle contains the following artifacts with the exception of the distro packages:

- DKP Kubernetes packages
- Python packages (provided by upstream)
- Containerd tarball

1. Download (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz && cd dkp-v2.8.1/kib
   ```

2. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.
3. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle` . This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml` . Example command:

244 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile
   ```

```
./konvoy-image create-package-bundle --os redhat-8.4 --output-
directory=artifacts
```

NOTE: For FIPS, pass the flag: `--fips`

NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY` . Example command:

```
export RHSM_ACTIVATION_KEY="-ci"
export RHSM_ORG_ID="1232131"
```

4.10.7.2.1 Setup Process:

1. The bootstrap image must be extracted and loaded onto the bastion host (see page 1068).
2. Artifacts must be copied onto cluster hosts for nodes to access.
3. If using GPU, those artifacts must be positioned locally.
4. Registry seeded with images locally.

4.10.7.2.2 Load the Bootstrap Image

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` from the download site (see page 76) mentioned above and extracted the tarball, you will load the bootstrap.
2. Load the bootstrap image on your bastion machine:

```
docker load -i konvoy-bootstrap-image-v2.8.1.tar
```

4.10.7.2.3 Copy air-gapped artifacts onto cluster hosts

Using the Konvoy Image Builder (see page 1626), you can copy the required artifacts onto your cluster hosts.

Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz` , continue below:

1. The Kubernetes image bundle will be located in `kib/artifacts/images` and you will want to verify image and artifacts.
   a. Verify the image bundles exist in `artifacts/images` :

```
$ ls artifacts/images/
```

```
kubernetes-images-1.28.7-d2iq.1.tar kubernetes-images-1.28.7-d2iq.1-
fips.tar
```

b. Verify the artifacts for your OS exist in the `artifacts/` directory and export the appropriate variables:

```
$ ls artifacts/
1.28.7_centos_7_x86_64.tar.gz       1.28.7_redhat_8_x86_64_fips.tar.gz
containerd-1.6.28-d2iq.1-rhel-7.9-x86_64.tar.gz   containerd-1.6.28-
d2iq.1-rhel-8.6-x86_64_fips.tar.gz  pip-packages.tar.gz
1.28.7_centos_7_x86_64_fips.tar.gz  1.28.7_rocky_9_x86_64.tar.gz
containerd-1.6.28-d2iq.1-rhel-7.9-x86_64_fips.tar.gz  containerd-1.6.28-
d2iq.1-rocky-9.0-x86_64.tar.gz
1.28.7_redhat_7_x86_64.tar.gz       1.28.7_ubuntu_20_x86_64.tar.gz
containerd-1.6.28-d2iq.1-rhel-8.4-x86_64.tar.gz   containerd-1.6.28-
d2iq.1-rocky-9.1-x86_64.tar.gz
1.28.7_redhat_7_x86_64_fips.tar.gz  containerd-1.6.28-d2iq.1-centos-7.9-
x86_64.tar.gz       containerd-1.6.28-d2iq.1-rhel-8.4-x86_64_fips.tar.gz
containerd-1.6.28-d2iq.1-ubuntu-20.04-x86_64.tar.gz  NVIDIA-Linux-
x86_64-470.82.01.run
1.28.7_redhat_8_x86_64.tar.gz       containerd-1.6.28-d2iq.1-centos-7.9-
x86_64_fips.tar.gz  containerd-1.6.28-d2iq.1-rhel-8.6-x86_64.tar.gz
images
```

c. For example, for RHEL 8.4 you would set:

```
export OS_PACKAGES_BUNDLE=1.28.7_redhat_8_x86_64.tar.gz
export CONTAINERD_BUNDLE=containerd-1.6.28-d2iq.1-rhel-8.4-x86_64.tar.gz
```

2. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_FILE="<private key file>"
```

`SSH_PRIVATE_KEY_FILE` must be either the name of the SSH private key file in your working directory or an absolute path to the file in your user's home directory.

3. Generate an `inventory.yaml` for GPU nodes which is automatically picked up by the `konvoy-image upload` in the next step.

```
cat <<EOF >  gpu_inventory.yaml
all:
  vars:
    ansible_port: 22
    ansible_ssh_private_key_file: $SSH_PRIVATE_KEY_FILE
    ansible_user: $SSH_USER

  hosts:
    $GPU_WORKER_1_ADDRESS:
      ansible_host: $GPU_WORKER_1_ADDRESS
EOF

cat <<EOF > inventory.yaml
all:
  vars:
    ansible_user: $SSH_USER
    ansible_port: 22
    ansible_ssh_private_key_file: $SSH_PRIVATE_KEY_FILE
  hosts:
    $CONTROL_PLANE_1_ADDRESS:
      ansible_host: $CONTROL_PLANE_1_ADDRESS
    $CONTROL_PLANE_2_ADDRESS:
      ansible_host: $CONTROL_PLANE_2_ADDRESS
    $CONTROL_PLANE_3_ADDRESS:
      ansible_host: $CONTROL_PLANE_3_ADDRESS
    $WORKER_1_ADDRESS:
      ansible_host: $WORKER_1_ADDRESS
    $WORKER_2_ADDRESS:
      ansible_host: $WORKER_2_ADDRESS
    $WORKER_3_ADDRESS:
      ansible_host: $WORKER_3_ADDRESS
    $WORKER_4_ADDRESS:
      ansible_host: $WORKER_4_ADDRESS
EOF
```

4. Upload the artifacts to the gpu nodepool with the `nvidia-runfile` flag with the following command:

```
konvoy-image upload artifacts --inventory-file=gpu_inventory.yaml \
              --container-images-dir=./artifacts/images/ \
              --os-packages-bundle=./artifacts/$OS_PACKAGES_BUNDLE \
              --containerd-bundle=artifacts/$CONTAINERD_BUNDLE \
              --pip-packages-bundle=./artifacts/pip-packages.tar.gz \
              --nvidia-runfile=./artifacts/NVIDIA-Linux-x86_64-470.82.01.run
```

KIB uses variable overrides (see page 1670) to specify base image and container images to use in your new machine image. The variable overrides files for NVIDIA and FIPS can be ignored unless adding an overlay feature.

- Use the `--overrides overrides/fips.yaml,overrides/offline-fips.yaml` flag with manifests located in the overrides directory[245] or see these pages in the documentation:
  - FIPS Overrides (see page 1671)
  - Create FIPS 140 Images (see page 1601)

4.10.7.2.3.1 Next Step:

Pre-provisioned Air-gapped GPU: Load the Registry (see page 267)

4.10.7.3 Pre-provisioned Air-gapped GPU: Load the Registry

Before creating an air-gapped Kubernetes cluster, you need to load the required images in a local registry for the Konvoy component. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1606) page for more information.

1. If not already done in prerequisites, download (see page 76) the air-gapped bundle `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap cluster, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

   ```
   cd dkp-v2.8.1
   ```

3. Set an environment variable with your registry address and any other needed variables using this command:

245 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
export REGISTRY_CA=<path to the cacert file on the bastion>
```

4. Execute the following command to load the air-gapped image bundle into your private registry using any of the relevant flags to apply variables above:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --
to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-
registry-password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

To increase Docker Hub's rate limit[246] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

4.10.7.3.1 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

4.10.7.3.2 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

246 https://docs.docker.com/docker-hub/download-rate-limit/

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

4.10.7.3.3 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME}
--to-registry-password=${REGISTRY_PASSWORD}
```

4.10.7.3.4 Next Step:

Pre-provisioned Air-gapped GPU: Nodepool Secrets and Overrides (see page 269)

4.10.7.4 Pre-provisioned Air-gapped GPU: Nodepool Secrets and Overrides

Install NVIDIA runfile and place it in the artifacts directory

For pre-provisioned environments, DKP has introduced the `nvidia-runfile` flag for Air-gapped Pre-provisioned environments. If the NVIDIA runfile[247] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory (you must create an `artifacts` directory if it doesn't already exist).

```
curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/NVIDIA-Linux-
x86_64-470.82.01.run
mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
```

DKP supported NVIDIA driver[248] version is 470.x.

247 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile
248 https://www.nvidia.com/Download/Find.aspx

1. Create the secret that GPU nodepool would use, this secret is populated from the KIB overrides. In this example we have a file called, `overrides/nvidia.yaml`. It should resemble this:

```
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
```

2. Create a secret on the bootstrap cluster that is populated from the above file. We will name it `${CLUSTER_NAME}-user-overrides`

```
kubectl create secret generic ${CLUSTER_NAME}-user-overrides --from-
file=overrides.yaml=overrides/nvidia.yaml
```

3. Create an inventory and nodepool with the instructions below and use the `$CLUSTER_NAME-user-overrides` secret.

Follow these steps:

1. Create an inventory object that has the same name as the node pool you're creating, and the details of the pre-provisioned machines that you want to add to it. For example, to create a node pool named `gpu-nodepool` an inventory named `gpu-nodepool` must be present in the same namespace:

```
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: ${MY_NODEPOOL_NAME}
spec:
  hosts:
    - address: ${IP_OF_NODE}
  sshConfig:
    port: 22
    user: ${SSH_USERNAME}
    privateKeyRef:
      name: ${NAME_OF_SSH_SECRET}
      namespace: ${NAMESPACE_OF_SSH_SECRET}
```

2. (Optional) If your pre-provisioned machines have overrides (see page 1670), you must create a secret that includes all of the overrides you want to provide in one file. Create an override secret (see page 1086) using the instructions detailed on this page.

3. Once the `PreprovisionedInventory` object and overrides are created, create a node pool:

```
dkp create nodepool preprovisioned -c ${MY_CLUSTER_NAME} ${MY_NODEPOOL_NAME} --
override-secret-name ${MY_OVERRIDE_SECRET}
```

- Advanced users can use a combination of the `--dry-run` and `--output=yaml` or `--output-directory=<existing-directory>` flags to get a complete set of node pool objects to modify locally or store in version control.

> For more information regarding this flag or others, please refer to the dkp create nodepool (see page 1875) section of the documentation for either cluster or nodepool and select your provider.

For more information, see:

- Pre-provisioned Create and Delete Node Pools (see page 1142)
- Pre-provisioned Air-gapped Define Environment (see page 1120)

Next Step:

Pre-provisioned Air-gapped GPU: Define Infrastructure (see page 271)

4.10.7.5 Pre-provisioned Air-gapped GPU: Define Infrastructure

Define the cluster hosts and infrastructure

Konvoy needs to know how to access your cluster hosts. This is done using inventory resources. For initial cluster creation, you must define a control-plane and at least one worker pool.

4.10.7.5.1 Define your Infrastructure

1. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_SECRET_NAME="$CLUSTER_NAME-ssh-key"
```

2. Use the following template to help you define your infrastructure. The environment variables that you set in the previous step automatically replace the variable names when the inventory YAML file is created.

```
cat <<EOF > preprovisioned_inventory.yaml
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-control-plane
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    # Create as many of these as needed to match your infrastructure
    # Note that the command line parameter --control-plane-replicas determines
how many control plane nodes will actually be used.
    #
    - address: $CONTROL_PLANE_1_ADDRESS
    - address: $CONTROL_PLANE_2_ADDRESS
    - address: $CONTROL_PLANE_3_ADDRESS
  sshConfig:
    port: 22
    # This is the username used to connect to your infrastructure. This user
must be root or
    # have the ability to use sudo without a password
    user: $SSH_USER
    privateKeyRef:
      # This is the name of the secret you created in the previous step. It
must exist in the same
      # namespace as this inventory object.
      name: $SSH_PRIVATE_KEY_SECRET_NAME
      namespace: default
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: $CLUSTER_NAME-md-0
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: $CLUSTER_NAME
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    - address: $WORKER_1_ADDRESS
    - address: $WORKER_2_ADDRESS
    - address: $WORKER_3_ADDRESS
    - address: $WORKER_4_ADDRESS
  sshConfig:
    port: 22
    user: $SSH_USER
    privateKeyRef:
      name: $SSH_PRIVATE_KEY_SECRET_NAME
```

```
      namespace: default
EOF
```

4.10.7.5.2 Next Step:

Pre-provisioned Air-gapped GPU: Define Control Plane Endpoint (see page 273)

4.10.7.6 Pre-provisioned Air-gapped GPU: Define Control Plane Endpoint

4.10.7.6.1 Define the Control Plane Endpoint for your cluster as well as connection mechanism

A control plane should have three, five, or seven nodes, so it can remain available if one or more nodes fail. A control plane with one node should not be used in production.

In addition, the control plane should have an endpoint that remains available if some nodes fail.

```
                        -------- cp1.example.com:6443
                        |
lb.example.com:6443 ---------- cp2.example.com:6443
                        |
                        -------- cp3.example.com:6443
```

In this example, the control plane endpoint host is `lb.example.com`, and the control plane endpoint port is `6443`. The control plane nodes are `cp1.example.com`, `cp2.example.com`, and `cp3.example.com`. The port of each API server is `6443`.

4.10.7.6.2 Select your Connection Mechanism

A virtual IP is the address that the client uses to connect to the service. A load balancer is the device that distributes the client connections to the backend servers. Before you create a new DKP cluster, choose an external load balancer(LB) or virtual IP.

- **External load balancer**

It is recommended that an external load balancer be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests.

- **Built-in virtual IP**

If an external load balancer is not available, use the built-in virtual IP (see page 1100). The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine.

4.10.7.6.3 Single-Node Control Plane

Do not use a single-node control plane in a production cluster.

A control plane with one node can use its single node as the endpoint, so you will not require an external load balancer, or a built-in virtual IP. At least one control plane node must always be running. Therefore, to upgrade a cluster with one control plane node, a spare machine must be available in the control plane inventory. This machine is used to provision the new node before the old node is deleted.

Modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1613).

When the API server endpoints are defined, you can create the cluster using the link in Next Step below.

4.10.7.6.4 Known Limitations

Be aware of these limitations in the current release of DKP.

The control plane endpoint port is also used as the API server port on each control plane machine. The default port is 6443. Before you create the cluster, ensure the port is available for use on each control plane machine.

4.10.7.6.5 Next Step:

Pre-provisioned Air-gapped GPU: Create a Cluster (see page 274)

4.10.7.7 Pre-provisioned Air-gapped GPU: Create a Management Cluster

If your cluster is air-gapped or you have a local registry (see page 1606), you must provide additional arguments when creating the cluster. These tell the cluster where to locate the local registry to use by defining the URL.

```
export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
export REGISTRY_CA=<path to the CA on the bastion>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

- `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.

4.10.7.7.1 Create a Cluster with GPU AMI

If a custom AMI was created using Konvoy Image Builder, the custom `ami` id is printed and written to `packer.pkr.hcl` .

To use the built `ami` with Konvoy, specify it with the `--ami` flag when calling cluster create in Step 1.

> For GPU Steps in Pre-provisioned (see page 0) section of the documentation to use the overrides/nvidia.yaml.

For GPU Steps in Pre-provisioned (see page 0) section of the documentation to use the overrides/nvidia.yaml.

Additional helpful information can be found in the NVIDIA Device Plug-in[249] for Kubernetes instructions and the Installation Guide of Supported Platforms[250].

> Before you create a new DKP cluster below, choose an external load balancer or virtual IP (see page 1100) and use the corresponding `dkp create cluster` command.

4.10.7.7.2 Name Your Cluster

> The cluster name may only contain the following characters: `a-z` , `0-9` , `.` , and `-` . Cluster creation will fail if the name has capital letters. See Kubernetes[251] for more naming information.

249 https://github.com/NVIDIA/k8s-device-plugin/blob/master/README.md
250 https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html
251 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-example>
```

4.10.7.7.3 Create an Air-gapped Kubernetes Cluster

After you have defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

DKP uses local static provisioner as the default storage provider (see page 110) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[252] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

For Pre-provisioned environments, you define a set of nodes that already exist. During the cluster creation process, Konvoy Image Builder (see page 1626)(KIB) is built into DKP and automatically runs the machine configuration process (which KIB uses to build images for other providers) against the set of nodes that you defined. This results in your pre-existing or **pre-provisioned** nodes being configured properly.

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory.

252 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

The create cluster command below includes the `--self-managed` flag. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

1. Execute this command to create a cluster with a GPU AMI using the default external load balancer option:

```
dkp create cluster preprovisioned \
  --cluster-name=${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --ami <ami> \
  --self-managed
```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

2. **Virtual IP ALTERNATIVE** - if you don't have an external LB, and wish to use a VIRTUAL IP (see page 1100) provided by kube-vip, specify these flags example below::

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --control-plane-endpoint-host 196.168.1.10 \
    --virtual-ip-interface eth1
```

   The output from this command is shortened here for reading clarity, but should start like this:

```
Generating cluster resources
cluster.cluster.x-k8s.io/preprovisioned-example created
cont.........
```

3. Create the node pool after cluster creation:

```
dkp create nodepool aws -c ${CLUSTER_NAME} \
  --instance-type p2.xlarge \
  --ami-id=${AMI_ID_FROM_KIB} \
```

```
  --replicas=1 ${NODEPOOL_NAME} \
  --kubeconfig=${CLUSTER_NAME}.conf
```

4. Use the wait command to monitor the cluster control-plane readiness:Use the wait command to monitor the cluster control-plane readiness:

   ```
   kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
   timeout=30m
   ```

   Output:

   ```
   cluster.cluster.x-k8s.io/preprovisioned-example condition met
   ```

> NOTE: Depending on the cluster size, it will take a few minutes to create.

When the command completes, you will have a running Kubernetes cluster! For bootstrap and custom YAML cluster creation, refer to the Custom Installation and Additional Infrastructure Tools (see page 1050) section of the documentation for Pre-provisioned: Pre-provisioned Infrastructure (see page 1070)

Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

> If changing the Calico encapsulation (see page 1095), D2iQ recommends changing it after cluster creation, but before production.

4.10.7.7.4 Audit logs

To modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1613).

4.10.7.7.4.1 Further Steps:

- Modify the Calico installation[253]
- Use the built-in Virtual IP[254]
- Provision on the Flatcar Linux OS[255]
- Use an HTTP proxy[256]
- Use alternate pod or service subnets[257]

1. **Virtual IP ALTERNATIVE** - if you don't have an external LB, and wish to use a VIRTUAL IP (see page 1100) provided by kube-vip, specify these flags example below:

   ```
   dkp create cluster preprovisioned \
       --cluster-name ${CLUSTER_NAME} \
       --control-plane-endpoint-host 196.168.1.10 \
       --virtual-ip-interface eth1
   ```

   The output from this command is shortened here for reading clarity, but should start like this:

   ```
   Generating cluster resources
   cluster.cluster.x-k8s.io/preprovisioned-example created
   cont.........
   ```

2. Use the wait command to monitor the cluster control-plane readiness:

   ```
   kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
   timeout=30m
   ```

   Output:

   ```
   cluster.cluster.x-k8s.io/preprovisioned-example condition met
   ```

NOTE: Depending on the cluster size, it will take a few minutes to create.

253 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/edit-v2/165347706#
254 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/edit-v2/165347706#
255 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/edit-v2/165347706#
256 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/edit-v2/165347706#
257 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/edit-v2/165347706#

When the command completes, you will have a running Kubernetes cluster! For bootstrap and custom YAML cluster creation, refer to the Additional Infrastructure Customization section of the documentation for Pre-provisioned: Pre-provisioned Infrastructure (see page 1070)

Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

If changing the Calico encapsulation (see page 1095), D2iQ recommends changing it after cluster creation, but before production.

4.10.7.7.4.2 Audit Logs

To modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1613).

4.10.7.7.4.3 Further Steps:

For more customized cluster creation, see the Pre-Provisioned Additional Configurations (see page 1070) section.

NOTE: The section mentioned above is for overrides for your clusters (see page 1680), custom flags, and more that specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied.

4.10.7.7.4.4 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.10.7.7.4.5 Next Step:

Pre-provisioned Air-gapped GPU: Configure MetalLB (see page 281)

4.10.7.8 Pre-provisioned Air-gapped GPU: Configure MetalLB

4.10.7.8.1 Create a MetalLB configmap for your pre-provisioned infrastructure.

Choose one of the following two protocols you want to use to announce service IPs. If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can continue the installation process.

To use MetalLB, create a MetalLB configMap for your Pre-provisioned infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

4.10.7.8.2 Layer 2 configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.10.7.8.3 BGP configuration

For a basic configuration featuring one BGP router and one IP address range, you need 4 pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

Once complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.10.7.8.3.1 Next Step:

Pre-provisioned Air-gapped GPU: Install Kommander (see page 283)

4.10.7.9 Pre-provisioned Air-gapped GPU: Install Kommander

4.10.7.9.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry: Air-gapped Environments (see page 1536).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

4.10.7.9.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init --airgapped > kommander.yaml
   ```

4. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP's default configuration ships Ceph with PVC based storage[258] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner (see page 110), you can install Ceph in host storage[259] mode.

   You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

   a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

      ```
      rook-ceph-cluster:
          enabled: true
      ```

258 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
259 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/

```
    values: |
      cephClusterSpec:
        storage:
          storageClassDeviceSets: []
          useAllDevices: true
          useAllNodes: true
          deviceFilter: "<<value>>"
```

b. To assign specific storage devices on all nodes to the Ceph cluster:

```
rook-ceph-cluster:
    enabled: true
    values: |
      cephClusterSpec:
        storage:
          storageClassDeviceSets: []
          useAllNodes: true
          useAllDevices: false
          deviceFilter: "^sdb."
```

Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[260]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[261].

5. *If required:* Customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, etc.

4.10.7.9.1.2 Enable GPU Resources

1. In the same `kommander.yaml` file, enable Nvidia platform services.

```
apps:
  nvidia-gpu-operator:
   enabled: true
```

2. Append the correct Toolkit version based on your OS:

   The **NVIDIA Container Toolkit** allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

   Centos 7.9/RHEL 7.9:
   If you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set

260 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
261 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.14.6-centos7
```

RHEL 8.4/8.6 and SLES 15 SP3

If you're using RHEL 8.4/8.6 or SLES 15 SP3 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.14.6-ubi8
```

Ubuntu 18.04 and 20.04

If you're using Ubuntu 18.04 or 20.04 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.14.6-ubuntu20.04
```

4.10.7.9.1.3 Enable DKP Catalog Applications and Install Kommander in an Air-gapped Environment

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add the following values to enable DKP Catalog Applications:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./dkp-catalog-applications-v2.8.1.tar.gz
```

⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.1.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.1.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.8.1.tar.gz
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).

- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.10.7.9.1.4 Next Step:

Pre-provisioned Air-gapped GPU: Verify Install and Log in to UI (see page 287)

4.10.7.10 Pre-provisioned Air-gapped GPU: Verify Install and Log in to UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.10.7.10.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

4.10.7.10.2 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

For more GPU information (see page 998), see the section for GPU under Day 2 - Cluster Operations Management.

4.10.7.10.3 Next Step:

Pre-provisioned Air-gapped with GPU: Subsequent Clusters (see page 289)

4.10.7.11 Pre-provisioned Air-gapped with GPU: Create Managed Clusters Using the DKP CLI

After initial cluster creation, you have the ability to create additional clusters from the CLI. In a previous step, the new cluster was created as Self-managed which allows it to be a Management cluster or a stand alone cluster. Subsequent new clusters are not self-managed as they will likely be Managed or Attached clusters to this Management Cluster.

When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

4.10.7.11.1 Make New Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.10.7.11.2 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[262] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-additional>
```

262 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

4.10.7.11.3 Create a Cluster with GPU AMI

If a custom AMI was created using Konvoy Image Builder, the custom `ami` id is printed and written to `./manifest.json`. To use the built `ami` with Konvoy, specify it with the `--ami` flag when calling cluster create in Step 1 in the next section where you create your Kubernetes cluster.

4.10.7.11.4 Create a Kubernetes Cluster

After you have defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster. This process creates a self-managed cluster to be used as the Management cluster.

Before you create a new DKP cluster below, choose an external load balancer (LB) or virtual IP (see page 1100) and use the corresponding `dkp create cluster` command.

In a pre-provisioned environment, use the Kubernetes CSI and third party drivers for local volumes and other storage devices in your data center.

DKP uses local static provisioner as the default storage provider (see page 110) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[263] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

For Pre-provisioned environments, you define a set of nodes that already exist. During the cluster creation process, Konvoy Image Builder (see page 1626)(KIB) is built into DKP and automatically runs the machine configuration process (which KIB uses to build images for other providers) against the set of nodes that you defined. This results in your pre-existing or **pre-provisioned** nodes being configured properly.

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory YAML (see page 149) previously created.

1. This command uses the default external load balancer (LB) option:

```
dkp create cluster preprovisioned \
```

263 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

```
  --cluster-name ${CLUSTER_NAME} \
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml \
  --ssh-private-key-file <path-to-ssh-private-key> \
  --registry-mirror-url=${_REGISTRY_URL} \
  --registry-mirror-cacert=${_REGISTRY_CA} \
  --registry-mirror-username=${_REGISTRY_USERNAME} \
  --registry-mirror-password=${_REGISTRY_PASSWORD}
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

2. Use the wait command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=30m
```

Output:

```
cluster.cluster.x-k8s.io/preprovisioned-additional condition met
```

NOTE: Depending on the cluster size, it will take a few minutes to create.

4.10.7.11.5 Manually Attach a DKP CLI Cluster to the Management Cluster

These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

```
echo ${MANAGED_CLUSTER_NAME}
```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
{MANAGED_CLUSTER_NAME}.conf
```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

```
kubectl get workspaces -A
```

5. Set the WORKSPACE_NAMESPACE environment variable:

```
export WORKSPACE_NAMESPACE=<workspace-namespace>
```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

```
kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
template='{{.data.value}}{{ "\n"}}'
```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

4.10.7.11.6 Next Step:

Day 2 - Cluster Operations Management (see page 590)

4.11 AWS Install Options

For an environment that is on the AWS Infrastructure, install options are provided for you in this one location.

Remember, there are always more options for custom YAML in the Custom Installation and Additional Infrastructure Tools (see page 1148) section, but this will get you operative in the most common scenarios.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

Below are all the supported environment combinations: (Refer to this page to check if your OS is a Supported Operating System (see page 67).)

- AWS Install (see page 295)
- AWS Air-gapped Install (see page 312)
- AWS with FIPS Install (see page 333)
- AWS Air-gapped FIPS Install (see page 350)

- AWS with GPU Install (see page 369)
- AWS Air-gapped with GPU Install (see page 389)

Additional Resource Information specific to AWS is below.

- **Control Plane Nodes** - DKP on AWS defaults to deploying an `m5.xlarge` instance with an 80GiB root volume for control plane nodes, which meets the above resource requirements (see page 119).
- **Worker Nodes** - DKP on AWS defaults to deploying a `m5.2xlarge` instance with an 80GiB root volume for worker nodes, which meets the above resource requirements (see page 119).

4.11.1 AWS Install

This section provides instructions to install DKP in an AWS non-air-gapped environment.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

4.11.1.1 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create permissions and roles on the Minimal Permissions and Role to Create Clusters (see page 1156) page.
2. Create Cluster IAM Policies and Roles (see page 1162).
3. Export the AWS region where you want to deploy the cluster:

   ```
   export AWS_REGION=us-west-2
   ```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

   ```
   export AWS_PROFILE=<profile>
   ```

If using AWS ECR as your local private registry, more information can be found on the Registry Mirror Tools (see page 1606) page.

To deploy a cluster with a custom image in a region where CAPI images[264] are not provided, you need to use Konvoy Image Builder (see page 1153) to create your own image for the region.

For multi-tenancy, every tenant should be in a different AWS account to ensure they are truly independent of other tenants in order to enforce security.

4.11.1.2 Next Step:

AWS: Create an Image (see page 296)

4.11.1.3 AWS: Create an Image

4.11.1.3.1 Learn how to build a custom AMI for use with DKP

AMI images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create an AMI image of your current computer system settings and software. The AMI image can then be replicated and distributed, creating your computer system for other users. You can use overrides files to customize some of the components installed on your machine image. For example, you could tell KIB to install the FIPS versions of the Kubernetes components.

This procedure describes how to use the Konvoy Image Builder (see page 1626) (KIB) to create a Cluster API[265] compliant Amazon Machine Image (AMI). KIB uses variable overrides (see page 1670) to specify base image and container images to use in your new AMI.

In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1629). Explore the Customize your Image (see page 1661) topic for more options about overrides.

264 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html
265 https://cluster-api.sigs.k8s.io/

4.11.1.3.2 Prerequisites

Before you begin, you must:

- Download the KIB (see page 0) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Check the Supported Infrastructure Operating Systems (see page 67)
- Check the Supported Kubernetes Version (see page 1706) for your Provider.
- Create a working registry:
    - Version 4.0 of Podman[266]or higher for Linux. Host requirements found here: https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
    - Version 20.10.0 of Docker[267] or higher for Linux or MacOS
- Ensure you have met the minimal set of permissions from the AWS Image Builder Book[268].
- A Minimal IAM Permissions for KIB (see page 1629) to create an Image for an AWS account using Konvoy Image Builder.

4.11.1.3.2.1 Extract KIB Bundle

If not done previously during Konvoy Image Builder download in Prerequisites, extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION` folder. **Otherwise, proceed to Build the Image.**

In previous DKP releases, the distro package bundles were included in the downloaded air-gapped bundle. Currently, that air-gapped bundle contains the following artifacts with the exception of the distro packages:

- DKP Kubernetes packages
- Python packages (provided by upstream)
- Containerd tarball

1. Download (see page 76) `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz` , and extract the tarball to a local directory:

    ```
    tar -xzvf dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz && cd dkp-v2.8.0/kib
    ```

2. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.
3. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle` . This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml` . Example command:

266 https://podman.io/getting-started/installation
267 https://www.docker.com/
268 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis

```
./konvoy-image create-package-bundle --os redhat-8.4 --output-
directory=artifacts
```

NOTE: For FIPS, pass the flag: `--fips`

NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY` . Example command:

```
export RHSM_ACTIVATION_KEY="-ci"
export RHSM_ORG_ID="1232131"
```

4. Follow the instructions below to build an AMI.

The `konvoy-image` binary and all supporting folders are also extracted. When run, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

- Set environment variables for AWS access[269]. The following variables must be set using your credentials including required IAM (see page 1629):

```
export AWS_ACCESS_KEY_ID
export AWS_SECRET_ACCESS_KEY
export AWS_DEFAULT_REGION
```

- If you have an override file (see page 1670) to configure specific attributes of your AMI file, add it. Instructions for customizing an override file are found on this page: Image Overrides (see page 1678)

4.11.1.3.2.2 Build the Image

Depending on which version of DKP (see page 1626) you are running, steps and flags will be different. To deploy in a region where CAPI images are not provided, you need to use KIB to create your own image for the region. For a list of supported AWS regions, refer to the Published AMI[270]information from AWS.

Begin image creation:

Run the `konvoy-image` command to build and validate the image.

```
konvoy-image build aws images/ami/rhel-86.yaml
```

269 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-envvars.html
270 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html

By default, it builds in the `us-west-2` region. to specify another region set the `--region` flag as shown in the command below:

```
konvoy-image build aws --region us-east-1 images/ami/rhel-86.yaml
```

Ensure you have named the correct AMI image YAML file for your OS[271] in the `konvoy-image build` command.

After KIB provisions the image successfully, the `ami` id is printed and written to the `packer.pkr.hcl` (Packer config) file. This file has an `artifact_id` field whose value provides the name of the AMI ID as shown in the example below. That is the `ami` you use in the `dkp create cluster` command:

```
{
  "builds": [
    {
      "name": "kib_image",
      "builder_type": "amazon-ebs",
      "build_time": 1698086886,
      "files": null,
      "artifact_id": "us-west-2:ami-04b8dfef8bd33a016",
      "packer_run_uuid": "80f8296c-e975-d394-45f9-49ef2ccc6e05",
      "custom_data": {
        "containerd_version": "",
        "distribution": "RHEL",
        "distribution_version": "8.6",
        "kubernetes_cni_version": "",
        "kubernetes_version": "1.26.6"
      }
    }
  ],
  "last_run_uuid": "80f8296c-e975-d394-45f9-49ef2ccc6e05"
}
```

To use a custom AMI when creating your cluster (see page 1190), you must create that AMI using KIB first. Then perform the export and name the custom AMI for use in the command `dkp create cluster`:

```
export AWS_AMI_ID=ami-<ami-id-here>
```

Inside the sections for either Non-air-gapped (see page 1190) or Air-gapped (see page 1218) cluster creation, you will find the instructions for how to apply custom images.

271 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ami

Related Information:

- To use a local registry even in a non-air-gapped environment, download and extract the bundle. Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.7.0_linux_amd64.tar.gz`) to load registry.
- Load the Registry (see page 317)

4.11.1.3.2.3 Next Step:

AWS: Create the Management Cluster (see page 300)

4.11.1.4 AWS: Create the Management Cluster

Use this procedure to create a self-managed AWS Management cluster with DKP. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

4.11.1.4.1 Name Your Cluster

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[272] for more naming information.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.
2. Set the environment variable:

```
export CLUSTER_NAME=<aws-example>
```

272 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

To increase Docker Hub's rate limit[273] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster` command.

4.11.1.4.2 Create a New AWS Kubernetes Cluster

If you use these instructions to create a cluster on AWS using the DKP default settings without any edits to configuration files or additional flags, your cluster is deployed on an Ubuntu 20.04 operating system image (see page 67) with 3 control plane nodes, and 4 worker nodes.

DKP uses AWS CSI as the default storage provider (see page 110). You can use a Kubernetes CSI[274] compatible storage solution that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[275] for more information.

In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1626).

There are two approaches to supplying the ID of your AMI. Either provide the ID of the AMI or provide a way for DKP to discover the AMI using location, format and OS information:

1. **Option One** - Provide the ID of your AMI:
    a. Use the example command below leaving the existing flag that provides the AMI ID: `--ami AMI_ID`
2. **Option Two** - Provide a path for your AMI with the information required for image discover:
    a. Where the AMI is published using your AWS Account ID: `--ami-owner AWS_ACCOUNT_ID`
    b. The format or string used to search for matching AMIs and ensure it references the Kubernetes version plus the base OS name: `--ami-base-os ubuntu-20.04`
    c. The base OS information: `--ami-format 'example-{{.BaseOS}}-?{{.K8sVersion}}-*'`

- ⚠ **IMPORTANT:** The AMI must be created with Konvoy Image Builder (see page 1626) in order to use the registry mirror feature.

```
export AWS_AMI_ID=<ami-...>
```

- **(Optional) Registry Mirror** - Configure your cluster to use an existing local registry as a mirror when attempting to pull images. Below is an AWS ECR example:

273 https://docs.docker.com/docker-hub/download-rate-limit/
274 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
275 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

```
export REGISTRY_URL=<ecr-registry-URI>
```

- `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.

1. Run this command to create your Kubernetes cluster using any relevant flags:

```
dkp create cluster aws \
--cluster-name=${CLUSTER_NAME} \
--ami=${AWS_AMI_ID} \
--additional-tags=owner=$(whoami) \
--with-aws-bootstrap-credentials=true \
--self-managed
```

Optional REGISTRY flag:

1. If you chose the option of creating a registry mirror, add this flag to your `dkp create cluster` command:

```
--registry-mirror-url=${REGISTRY_URL}
```

- Other registry flag options to export and use:

```
export REGISTRY_URL=<registry-address>:<registry-port>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
export REGISTRY_CA=<path to the cacert file on the bastion>
```

```
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD}
```

2. Custom AMI flag:

```
--ami=${AWS_AMI_ID}
```

Flatcar OS

Flatcar OS (see page 1183) use `--os-hint` to instruct the bootstrap cluster to make some changes related to the installation paths:

```
--os-hint flatcar
```

> If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

4.11.1.4.3 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.11.1.4.4 Next Step:

AWS: Install Kommander (see page 303)

4.11.1.5 AWS: Install Kommander

You have installed the Konvoy component and started creating a Management cluster. Now it is time to Install Kommander which completes the Management cluster and allows you to access the UI and attach new or existing managed clusters to monitor.

4.11.1.5.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

4.11.1.5.1.1 Create and customize your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1558) for the deployment:

```
dkp install kommander --init > kommander.yaml
```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1541)), etc.
5. *If required:* If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true"
```

4.11.1.5.1.2 Enable DKP Catalog Applications and Install Kommander

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
```

```
      url: https://github.com/mesosphere/dkp-catalog-applications
      ref:
        tag: v2.7.0
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.11.1.5.1.3 Next Step:

AWS: Verify Install and Log in the UI (see page 305)

4.11.1.6 AWS: Verify Install and Log in the UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.11.1.6.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

4.11.1.6.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

4.11.1.6.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as

attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

Now you can create additional new clusters (see page 308), or proceed to Day 2 Cluster Operations (see page 590) for attaching existing cluster..

4.11.1.6.4 Next Step:

AWS: Create a Managed Cluster Using the DKP CLI (see page 308)

4.11.1.7 AWS: Create a Managed Cluster Using the DKP CLI

Enterprise Gov Advanced

In previous steps, you created a new Management cluster, which is self-managed. When you use these steps to create new Managed clusters, they will become Attached clusters under your Management Cluster.

> ⚠ When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

Your new, managed cluster needs to be part of either a new Workspace or an existing Workspace under a management cluster. Create or locate a workspace name to get started with your managed cluster.

4.11.1.7.1 Make New Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.11.1.7.1.1 Name Your Cluster

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[276] for more naming information.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

```
export MANAGED_CLUSTER_NAME=<aws-additional>
```

4.11.1.7.2 Create a New Cluster with the CLI

> The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.
> If you do not set a workspace, it will be created in the `default` workspace, and you need to take additional steps to attach to a workspace later. For instructions on how to do this, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster). (see page 784).

Execute this command to create your additional Kubernetes cluster using any relevant flags. This will create a new non-self-managed cluster that can be managed by management cluster (see page 300) you created in the previous section:

```
dkp create cluster aws \
--cluster-name=${MANAGED_CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
```

276 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
--kubeconfig=<management-cluster-kubeconfig-path> \
--namespace ${WORKSPACE_NAMESPACE}
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

4.11.1.7.3 Retrieve the kubeconfig and Explore New AWS Cluster

Follow these steps:

1. Fetch the kubeconfig file with the command:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} --kubeconfig
   <management-cluster-kubeconfig-path> -n ${WORKSPACE_NAMESPACE} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

2. List the nodes with the following command:

   ```
   kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get nodes
   ```

3. List the pods with the following command:

NOTE: Wait for the Status to move to Ready while `calico-node` pods are being deployed.

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get pods -A
```

4.11.1.7.3.1 Manually Attach a DKP CLI Cluster to the Management Cluster

These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

```
echo ${MANAGED_CLUSTER_NAME}
```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
{MANAGED_CLUSTER_NAME}.conf
```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

```
kubectl get workspaces -A
```

5. Set the WORKSPACE_NAMESPACE environment variable:

```
export WORKSPACE_NAMESPACE=<workspace-namespace>
```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

```
kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
template='{{.data.value}}{{ "\n"}}'
```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
```

```
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

If you have existing clusters or want to create new clusters to attach, there are many ways to attach a cluster with various requirements and restrictions. To see all the options, visit the section in documentation Day 2 - Attach an Existing Kubernetes Cluster (see page 784).

4.11.1.7.3.2 Next Step:

Day 2 - Cluster Operations Management (see page 590)

4.11.2 AWS Air-gapped Install

This section provides the instructions to install DKP in an AWS air-gapped environment.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)

- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

4.11.2.1 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create permissions and roles on the Minimal Permissions and Role to Create Clusters (see page 1156) page.
2. Create Cluster IAM Policies and Roles (see page 1162).
3. Export the AWS region where you want to deploy the cluster:

```
export AWS_REGION=us-west-2
```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

```
export AWS_PROFILE=<profile>
```

If using AWS ECR as your local private registry, more information can be found on the Registry Mirror Tools (see page 1606) page.

To deploy a cluster with a custom image in a region where CAPI images[277] are not provided, you need to use Konvoy Image Builder (see page 1153) to create your own image for the region.

For multi-tenancy, every tenant should be in a different AWS account to ensure they are truly independent of other tenants in order to enforce security.

For air-gapped, ensure you have downloaded (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , so you can extract the tarball on the page with those instructions.

277 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html

4.11.2.2 Next Step:

AWS Air-gapped: Create an Image (see page 314)

4.11.2.3 AWS Air-gapped: Create an Image

This procedure describes how to use the Konvoy Image Builder (see page 1626) (KIB) to create a Cluster API[278] compliant Amazon Machine Image (AMI). AMI images contain configuration information and software to create a specific, pre-configured, operating environment. KIB uses variable overrides (see page 1670) to specify base image and container images to use in your new AMI.

4.11.2.3.1 Create an Air-gapped AMI

To create an Image using Konvoy Image Builder (KIB) for use in an air-gapped cluster, follow these instructions. AMI images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create an AMI image of your current computer system settings and software. The AMI image can then be replicated and distributed, creating your computer system for other users. You can use overrides files to customize some of the components installed on your machine image. For example, you could tell KIB to install the FIPS versions of the Kubernetes components.

4.11.2.3.2 Prerequisites

- Minimal IAM Permissions for KIB (see page 1629) to create an Image for an AWS account using Konvoy Image Builder.

In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1626).
Explore the Customize your Image (see page 1661) topic for more options.
Using KIB (see page 1626), you can build an AMI without requiring access to the internet by providing an additional `--override` flag.

In previous DKP releases, the distro package bundles were included in the downloaded air-gapped bundle. Currently, that air-gapped bundle contains the following artifacts with the exception of the distro packages:

- DKP Kubernetes packages
- Python packages (provided by upstream)
- Containerd tarball

278 https://cluster-api.sigs.k8s.io/

1. Download (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz && cd dkp-v2.8.1/kib
```

2. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.
3. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle` . This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml` . Example command:

```
./konvoy-image create-package-bundle --os redhat-8.4 --output-
directory=artifacts
```

NOTE: For FIPS, pass the flag: `--fips`

NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY` . Example command:

```
export RHSM_ACTIVATION_KEY="-ci"
export RHSM_ORG_ID="1232131"
```

4. Follow the instructions below to build an AMI.

Depending on which version of DKP (see page 1626) you are running, steps and flags will be different. To deploy in a region where CAPI images are not provided, you need to use KIB to create your own image for the region. For a list of supported AWS regions, refer to the Published AMI[279]information from AWS.

4.11.2.3.2.1 Run the following to begin image creation:

5. Run the `konvoy-image` command to build and validate the image. Ensure you have named the correct AMI image YAML file for your OS[280] in the `konvoy-image build` command.

```
./konvoy-image build aws images/ami/centos-79.yaml --overrides overrides/
offline.yaml
```

By default, it builds in the `us-west-2` region. **To specify another region** set the `--region` flag as shown in the example command below:

279 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html
280 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ami

```
./konvoy-image build aws --region us-east-1 images/ami/centos-79.yaml --overrides
overrides/offline.yaml
```

Instructions for customizing an override file are found on this page: Image Overrides (see page 1678)

After KIB provisions the image successfully, the `ami` id is printed and written to the `packer.pkr.hcl` (Packer config) file. This file has an `amazon-ebs.kib_image` field whose value provides the name of the AMI ID as shown in the example below. That is the `ami` you use in the `dkp create cluster` command:

```
...
    amazon-ebs.kib_image: Adding tag: "distribution_version": "8.6"
    amazon-ebs.kib_image: Adding tag: "gpu_nvidia_version": ""
    amazon-ebs.kib_image: Adding tag: "kubernetes_cni_version": ""
    amazon-ebs.kib_image: Adding tag: "build_timestamp": "20231023182049"
    amazon-ebs.kib_image: Adding tag: "gpu_types": ""
    amazon-ebs.kib_image: Adding tag: "kubernetes_version": "1.28.7"
==> amazon-ebs.kib_image: Creating snapshot tags
    amazon-ebs.kib_image: Adding tag: "ami_name": "konvoy-ami-
rhel-8.6-1.26.6-20231023182049"
==> amazon-ebs.kib_image: Terminating the source AWS instance...
==> amazon-ebs.kib_image: Cleaning up any extra volumes...
==> amazon-ebs.kib_image: No volumes to clean up, skipping
==> amazon-ebs.kib_image: Deleting temporary security group...
==> amazon-ebs.kib_image: Deleting temporary keypair...
==> amazon-ebs.kib_image: Running post-processor:  (type manifest)
Build 'amazon-ebs.kib_image' finished after 26 minutes 52 seconds.

==> Wait completed after 26 minutes 52 seconds

==> Builds finished. The artifacts of successful builds are:
--> amazon-ebs.kib_image: AMIs were created:
us-west-2: ami-04b8dfef8bd33a016

--> amazon-ebs.kib_image: AMIs were created:
us-west-2: ami-04b8dfef8bd33a016
```

4.11.2.3.2.2 Next Step:

AWS Air-gapped: Docker Registry (see page 317)

4.11.2.4 AWS Air-gapped: Load the Registry

After you create an image for your air-gapped environment, you will need to load it into your registry.

4.11.2.4.1 Load Images into your Registry

Before creating an air-gapped Kubernetes cluster, you need to load the required images in a local registry for the Konvoy component. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1606) page for more information.

1. If not already done in prerequisites, download (see page 76) the air-gapped bundle `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap cluster, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

   ```
   cd dkp-v2.8.1
   ```

3. Set an environment variable with your registry address and any other needed variables using this command:

   ```
   export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   export REGISTRY_CA=<path to the cacert file on the bastion>
   ```

4. Execute the following command to load the air-gapped image bundle into your private registry using any of the relevant flags to apply variables above:

   ```
   dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --
   to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-
   registry-password=${REGISTRY_PASSWORD}
   ```

> It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

4.11.2.4.1.1 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

4.11.2.4.1.2 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

4.11.2.4.1.3 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME}
--to-registry-password=${REGISTRY_PASSWORD}
```

4.11.2.4.1.4 Next Step:

AWS Air-gapped: Create the Management Cluster (see page 318)

4.11.2.5 AWS Air-gapped: Create the Management Cluster

Create a new self-managed AWS Kubernetes cluster in an Air-gapped environment on your AWS infrastructure. A self-managed cluster refers to one in which the CAPI resources and controllers that

describe and manage it are running on the same cluster they are managing. When you use existing infrastructure, DKP does *not* create, modify, or delete the following AWS resources:

- Internet Gateways
- NAT Gateways
- Routing tables
- Subnets
- VPC
- VPC Endpoints (for subnets without NAT Gateways)

An AWS subnet has Network ACLs that can control traffic in and out of the subnet. DKP does not modify the Network ACLs of an existing subnet. DKP uses Security Groups to control traffic. If a Network ACL denies traffic that is allowed by DKP-managed Security Groups, the cluster may not work correctly.

To increase Docker Hub's rate limit[281] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster` command.

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable to the name you assigned this cluster:

   ```
   export CLUSTER_NAME=<aws-example>
   ```

   NOTE: The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[282] for more naming information.

3. Export variables for the existing infrastructure details:

   ```
   export AWS_VPC_ID=<vpc-...>
   export AWS_SUBNET_IDS=<subnet-...,subnet-...,subnet-...>
   export AWS_ADDITIONAL_SECURITY_GROUPS=<sg-...>
   ```

281 https://docs.docker.com/docker-hub/download-rate-limit/
282 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
export AWS_AMI_ID=<ami-...>
```

- `AWS_VPC_ID` : the VPC ID where the cluster will be created. The VPC requires the following AWS VPC Endpoints to be already present:
  - `ec2` - `com.amazonaws.{region}.ec2`
  - `elasticloadbalancing` - `com.amazonaws.{region}.elasticloadbalancing`
  - `secretsmanager` - `com.amazonaws.{region}.secretsmanager`
  - `autoscaling` - `com.amazonaws.{region}.autoscaling`
  - `ecr` - `com.amazonaws.{region}.ecr.api` - `(authentication)`
  - `ecr` - `com.amazonaws.{region}.ecr.dkr` - `(data trasfer)`

  More details about AWS service using an interface VPC endpoint[283] and AWS VPC endpoints list[284].
- `AWS_SUBNET_IDS` : a comma-separated list of one or more private Subnet IDs with each one in a different Availability Zone. The cluster control-plane and worker nodes will automatically be spread across these Subnets.
- `AWS_ADDITIONAL_SECURITY_GROUPS` : a comma-seperated list of one or more Security Groups IDs to use in addition to the ones automatically created by CAPA[285].
- `AWS_AMI_ID` : the AMI ID to use for control-plane and worker nodes. The AMI must be created by the konvoy-image-builder (see page 1626).

In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1626).

There are two approaches to supplying the ID of your AMI while creating your cluster.

- **Option One** is to provide the ID of the AMI: `--ami AMI_ID` .
- **Option Two** is provide a way for DKP to discover the AMI using location, format and OS information using flags: AWS Account ID: `--ami-owner AWS_ACCOUNT_ID` , format `--ami-base-os ubuntu-20.04` , and OS `--ami-format 'example-{{.BaseOS}}-?{{.K8sVersion}}-*'` . Examples of these choices are shown in the `dkp create cluster aws` code snippets below.

283 https://docs.aws.amazon.com/vpc/latest/privatelink/create-interface-endpoint.html
284 https://docs.aws.amazon.com/vpc/latest/privatelink/aws-services-privatelink-support.html
285 https://github.com/kubernetes-sigs/cluster-api-provider-aws

4. ⚠ **IMPORTANT:** You must tag the subnets as described below to allow for Kubernetes to create External Load Balancers (ELBs) for services of type `LoadBalancer` in those subnets. If the subnets are not tagged, they will not receive an ELB and the following error displays: `Error syncing load balancer, failed to ensure load balancer; could not find any suitable subnets for creating the ELB.`.

   The tags should be set as follows, where `<CLUSTER_NAME>` corresponds to the name set in `CLUSTER_NAME` environment variable:

   ```
   kubernetes.io/cluster = <CLUSTER_NAME>
   kubernetes.io/cluster/<CLUSTER_NAME> = owned
   kubernetes.io/role/internal-elb = 1
   ```

5. (Optional) Configure your cluster to use an existing container registry as a mirror when attempting to pull images. The example below is for AWS ECR:
   ⚠ If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1606) page for more information.

   ⚠ **IMPORTANT:** The AMI must be created by the konvoy-image-builder (see page 1626) project in order to use the registry mirror feature.

   ```
   export REGISTRY_URL=<ecr-registry-URI>
   ```

   - `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
   - NOTE: Other local registries may use the options below:
     - JFrog - `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
     - `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
     - `REGISTRY_PASSWORD` : optional if username is not set.

6. Create a Kubernetes cluster. The following example shows a common configuration. See dkp create cluster aws (see page 1856) reference for the full list of cluster creation options:

⚠ DKP uses AWS CSI as the default storage provider (see page 110). You can use a Kubernetes CSI[286] compatible storage solution that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[287] for more information.

286 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
287 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

1. Run this command to create your Kubernetes cluster using any relevant flags for **Option One** explained above providing the AMI ID:

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--with-aws-bootstrap-credentials=true \
--vpc-id=${AWS_VPC_ID} \
--ami=${AWS_AMI_ID} \
--subnet-ids=${AWS_SUBNET_IDS} \
--internal-load-balancer=true \
--additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
--registry-mirror-url=<YOUR_ECR_URL> \
--self-managed
```

OR

2. **Option Two** is to run the command as shown from the explanation above to allow discovery of your AMI:

```
dkp create cluster aws \
--cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--with-aws-bootstrap-credentials=true \
--vpc-id=${AWS_VPC_ID} \
--ami-owner AWS_ACCOUNT_ID \
--ami-base-os ubuntu-20.04 \
--ami-format 'example-{{.BaseOS}}-?{{.K8sVersion}}-*' \
--subnet-ids=${AWS_SUBNET_IDS} \
--internal-load-balancer=true \
--additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
--registry-mirror-url=<YOUR_ECR_URL> \
--self-managed
```

Flatcar OS

Flatcar OS (see page 1183) use `--os-hint` to instruct the bootstrap cluster to make some changes related to the installation paths:

```
--os-hint flatcar
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

You can find a customizable YAML and other custom instructions in Create a new Air-gapped AWS Cluster[288] under Custom Installation and Additional Infrastructure Tools (see page 1050) .

4.11.2.5.1 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.11.2.5.2 Next Step:

AWS Air-gapped: Install Kommander (see page 323)

4.11.2.6 AWS Air-gapped: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

4.11.2.6.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry: Air-gapped Environments (see page 1536).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

4.11.2.6.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init --airgapped > kommander.yaml
   ```

288 https://d2iq.atlassian.net/wiki/pages/createpage.action?fromPageId=161186126&linkCreation=true&spaceKey=DENT&title=AWS+Cluster+Creation+Choices

4. *If required:* Customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1541)), etc.
5. *If required*: If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true"
```

4.11.2.6.1.2 Enable DKP Catalog Applications and Install Kommander in an Air-gapped Environment

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add the following values to enable DKP Catalog Applications:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./dkp-catalog-applications-v2.8.1.tar.gz
```

   If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.
2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.1.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.1.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.8.1.tar.gz
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.11.2.6.1.3 Next Step:

AWS Air-gapped: Verify Install and Log in the UI (see page 325)

4.11.2.7 AWS Air-gapped: Verify Install and Log in the UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.11.2.7.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
```

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

4.11.2.7.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

4.11.2.7.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2

section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

Now you can create additional new clusters (see page 328), or proceed to Day 2 Operations (see page 590).

4.11.2.7.4 Next Step:

AWS Air-gapped: Subsequent New Clusters (see page 328)

4.11.2.8 AWS Air-gapped: Create a Managed Cluster Using the DKP CLI

Enterprise Gov Advanced

In previous steps, you created a new Management cluster, which is self-managed. When you use these steps to create new Managed clusters, they will become Attached clusters under your Management Cluster.

When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

4.11.2.8.1 Make New Managed Cluster Part of a Workspace

Your new managed cluster needs to be part of either a new Workspace or an existing Workspace.

To create a new Workspace, follow the instructions to Create a New Workspace[289].

To find the existing Workspaces, use this command:

```
kubectl get workspace -A
```

When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

289 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/213320100

4.11.2.8.2 Name Your Cluster

> The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.
> If you do not set a workspace, it will be created in the `default` workspace, and you need to take additional steps to attach to a workspace later. For instructions on how to do this, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster). (see page 784).

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

```
export MANAGED_CLUSTER_NAME=<aws-additional>
```

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[290] for more naming information.

3. Execute this command to create your Kubernetes cluster using any relevant flags but note that the `--self-managed` is absent so that this cluster can be attached to your previously created Management Cluster.

```
dkp create cluster aws --cluster-name=${MANAGED_CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--namespace ${WORKSPACE_NAMESPACE}
--kubeconfig=<management-cluster-kubeconfig-path> \
--with-aws-bootstrap-credentials=true \
--vpc-id=${AWS_VPC_ID} \
--ami=${AWS_AMI_ID} \
--subnet-ids=${AWS_SUBNET_IDS} \
--internal-load-balancer=true \
```

290 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
--additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
--registry-mirror-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD} \
--kubeconfig=<management-cluster-kubeconfig-path> \
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

4.11.2.8.3 Retrieve the kubeconfig and Explore New AWS Cluster

Follow these steps:

1. Fetch the kubeconfig file with the command:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} --kubeconfig
   <management-cluster-kubeconfig-path> -n ${WORKSPACE_NAMESPACE} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

2. List the nodes with the following command:

   ```
   kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get nodes
   ```

3. List the pods with the following command:

NOTE: Wait for the Status to move to Ready while `calico-node` pods are being deployed.

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get pods -A
```

4.11.2.8.3.1 Manually Attach a DKP CLI Cluster to the Management Cluster

> ⚠ These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

   ```
   echo ${MANAGED_CLUSTER_NAME}
   ```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
   NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

   ```
   kubectl get workspaces -A
   ```

5. Set the WORKSPACE_NAMESPACE environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace-namespace>
   ```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

   ```
   kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
   template='{{.data.value}}{{ "\n"}}'
   ```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

4.11.2.8.3.2 Next Step:

Day 2 - Cluster Operations Management (see page 590)

4.11.3 AWS with FIPS Install

This section provides instructions to install DKP in an AWS non-air-gapped FIPS environment.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

4.11.3.1 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create permissions and roles on the Minimal Permissions and Role to Create Clusters (see page 1156) page.
2. Create Cluster IAM Policies and Roles (see page 1162).
3. Export the AWS region where you want to deploy the cluster:

   ```
   export AWS_REGION=us-west-2
   ```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

   ```
   export AWS_PROFILE=<profile>
   ```

If using AWS ECR as your local private registry, more information can be found on the Registry Mirror Tools (see page 1606) page.

To deploy a cluster with a custom image in a region where CAPI images[291] are not provided, you need to use Konvoy Image Builder (see page 1153) to create your own image for the region.

For multi-tenancy, every tenant should be in a different AWS account to ensure they are truly independent of other tenants in order to enforce security.

291 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html

4.11.3.2 Next Step:

AWS FIPS: Create an Image (see page 334)

4.11.3.3 AWS FIPS: Create an Image

4.11.3.3.1 Learn how to build a custom AMI for use with DKP

AMI images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create an AMI image of your current computer system settings and software. The AMI image can then be replicated and distributed, creating your computer system for other users. You can use overrides files to customize some of the components installed on your machine image. For example, you could tell KIB to install the FIPS versions of the Kubernetes components.

This procedure describes how to use the Konvoy Image Builder (see page 1626) (KIB) to create a Cluster API[292] compliant Amazon Machine Image (AMI). KIB uses variable overrides (see page 1670) to specify base image and container images to use in your new AMI.

In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1629). Explore the Customize your Image (see page 1661) topic for more options about overrides.

4.11.3.3.2 Prerequisites

Before you begin, you must:

- Download the KIB (see page 0) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Check the Supported Infrastructure Operating Systems (see page 67)
- Check the Supported Kubernetes Version (see page 1706) for your Provider.
- Create a working registry:
    - Version 4.0 of Podman[293]or higher for Linux. Host requirements found here: https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
    - Version 20.10.0 of Docker[294] or higher for Linux or MacOS

292 https://cluster-api.sigs.k8s.io/
293 https://podman.io/getting-started/installation
294 https://www.docker.com/

- Ensure you have met the minimal set of permissions from the AWS Image Builder Book[295].
- A Minimal IAM Permissions for KIB (see page 1629) to create an Image for an AWS account using Konvoy Image Builder.

4.11.3.3.2.1 Extract KIB Bundle

If not done previously during Konvoy Image Builder download in Prerequisites, extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION` folder. **Otherwise, proceed to Build the Image.**

In previous DKP releases, the distro package bundles were included in the downloaded air-gapped bundle. Currently, that air-gapped bundle contains the following artifacts with the exception of the distro packages:

- DKP Kubernetes packages
- Python packages (provided by upstream)
- Containerd tarball

1. Download (see page 76) `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz && cd dkp-v2.8.0/kib
   ```

2. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.
3. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle` . This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml` . Example command:

   ```
   ./konvoy-image create-package-bundle --os redhat-8.4 --output-
   directory=artifacts
   ```

   NOTE: For FIPS, pass the flag: `--fips`
   NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY` . Example command:

   ```
   export RHSM_ACTIVATION_KEY="-ci"
   export RHSM_ORG_ID="1232131"
   ```

4. Follow the instructions below to build an AMI.

295 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis

The `konvoy-image` binary and all supporting folders are also extracted. When run, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

- Set environment variables for AWS access[296]. The following variables must be set using your credentials including required IAM (see page 1629):

```
export AWS_ACCESS_KEY_ID
export AWS_SECRET_ACCESS_KEY
export AWS_DEFAULT_REGION
```

- If you have an override file (see page 1670) to configure specific attributes of your AMI file, add it. Instructions for customizing an override file are found on this page: Image Overrides (see page 1678)

4.11.3.3.2.2 Non-air-gapped Environment Create FIPS-140 images

KIB can produce images containing FIPS-140 compliant binaries. Use the `fips.yaml` override file (see page 1671) provided with the image bundles.

You can also find these override files in the Konvoy Image Builder repo[297].

- A non-air-gapped example of override file use is the command below, which produces a FIPS-compliant image on RHEL 8.4 for AWS:
  Replace `ami` with your infrastructure provisioner

```
konvoy-image build --overrides overrides/fips.yaml images/ami/rhel-84.yaml
```

Related Information:

- To use a local registry even in a non-air-gapped environment, download and extract the bundle. Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.7.0_linux_amd64.tar.gz`) to load registry.
- Load the Registry (see page 317)

296 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-envvars.html
297 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

4.11.3.3.2.3 Next Step:

AWS FIPS: Create the Management Cluster (see page 337)

4.11.3.4 AWS FIPS: Create the Management Cluster

Use this procedure to create a self-managed AWS FIPS Management cluster with DKP. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing. Additional considerations exist when using FIPS. Alter the Create a Kubernetes cluster command by using the FIPS example, or similar, when executing the command to create a cluster.

4.11.3.4.1 Deploying a Cluster in FIPS mode

In order to create a cluster in FIPS mode, we must inform the bootstrap controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes.

4.11.3.4.1.1 Supported FIPS Builds

Component	Repository	Version
Kubernetes	docker.io/mesosphere[298]	v1.28.7+fips.0
etcd	docker.io/mesosphere[299]	3.5.10+fips.0

4.11.3.4.1.2 Name Your Cluster

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[300] for more naming information.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

298 https://hub.docker.com/layers/mesosphere/kube-apiserver/v1.28.7_d2iq.0/images/sha256-1cafe40f5a17c86aa75574b25dd637bbeb377e2ac703532d72f1118c1e966e28?context=explore
299 https://hub.docker.com/layers/mesosphere/etcd/v3.5.10_fips.0/images/sha256-60da8d0d3b37e71a4fa918ffa7ffd9963d9892b3ba43b8f63119d806f2e585ed?context=explore
300 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

```
export CLUSTER_NAME=<aws-example>
```

4.11.3.4.1.3 Create a New AWS Kubernetes Cluster

If you use these instructions to create a cluster on AWS using the DKP default settings without any edits to configuration files or additional flags, your cluster is deployed on an Ubuntu 20.04 operating system image (see page 67) with 3 control plane nodes, and 4 worker nodes.

DKP uses AWS CSI as the default storage provider (see page 110). You can use a Kubernetes CSI[301] compatible storage solution that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[302] for more information.

In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1626).

There are two approaches to supplying the ID of your AMI. Either provide the ID of the AMI or provide a way for DKP to discover the AMI using location, format and OS information:

1. **Option One** - Provide the ID of your AMI:
   a. Use the example command below leaving the existing flag that provides the AMI ID: `--ami AMI_ID`
2. **Option Two** - Provide a path for your AMI with the information required for image discover:
   a. Where the AMI is published using your AWS Account ID: `--ami-owner AWS_ACCOUNT_ID`
   b. The format or string used to search for matching AMIs and ensure it references the Kubernetes version plus the base OS name: `--ami-base-os ubuntu-20.04`
   c. The base OS information: `--ami-format 'example-{{.BaseOS}}-?{{.K8sVersion}}-*'`

301 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
302 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

- ⚠️ **IMPORTANT:** The AMI must be created with Konvoy Image Builder (see page 1626) in order to use the registry mirror feature.

```
export AWS_AMI_ID=<ami-...>
```

- **(Optional) Registry Mirror** - Configure your cluster to use an existing local registry as a mirror when attempting to pull images. Below is an AWS ECR example:

```
export REGISTRY_URL=<ecr-registry-URI>
```

  - `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.

1. Run this command to create your Kubernetes cluster using any relevant flags for **Option One** explained above:

```
dkp create cluster aws \
--cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--with-aws-bootstrap-credentials=true \
--ami AMI_ID \
--kubernetes-version=v1.28.7+fips.0 \
--etcd-version=3.5.10+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
--self-managed
```

OR

2. **Option Two** is to run the command as shown from the explanation above:

```
dkp create cluster aws \
--cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--with-aws-bootstrap-credentials=true \
--ami-owner AWS_ACCOUNT_ID \
--ami-base-os ubuntu-20.04 \
--ami-format 'example-{{.BaseOS}}-?{{.K8sVersion}}-*' \
--kubernetes-version=v1.28.7+fips.0 \
--etcd-version=3.5.10+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
--self-managed
```

Flatcar OS

Flatcar OS (see page 1183) use `--os-hint` to instruct the bootstrap cluster to make some changes related to the installation paths:

```
--os-hint flatcar
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

You can find a customizable Create a new AWS Cluster[303] under the Custom Installation and Additional Infrastructure Tools (see page 1050). For more details about FIPS, refer to the FIPS 140-2 Compliance (see page 1598) section of the documentation.

4.11.3.4.1.4 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.11.3.4.1.5 Next Step:

AWS FIPS: Install Kommander (see page 340)

4.11.3.5 AWS FIPS: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

4.11.3.5.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

4.11.3.5.1.1 Create and customize your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

303 https://d2iq.atlassian.net/wiki/pages/createpage.action?fromPageId=161776565&linkCreation=true&spaceKey=DENT&title=AWS+Cluster+Creation+Choices

```
export CLUSTER_NAME=<your-management-cluster-name>
```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1558) for the deployment:

```
dkp install kommander --init > kommander.yaml
```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1541)), etc.

5. *If required:* If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true"
```

4.11.3.5.1.2 Enable DKP Catalog Applications and Install Kommander

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
```

```
  - name: dkp-catalog-applications
    labels:
      kommander.d2iq.io/project-default-catalog-repository: "true"
      kommander.d2iq.io/workspace-default-catalog-repository: "true"
      kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
    gitRepositorySpec:
      url: https://github.com/mesosphere/dkp-catalog-applications
      ref:
        tag: v2.7.0
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.11.3.5.1.3 Next Step:

AWS FIPS: Verify Install and Log in the UI (see page 342)

4.11.3.6 AWS FIPS: Verify Install and Log in the UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.11.3.6.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

4.11.3.6.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider

- Create Groups

4.11.3.6.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

Now you can create additional new clusters (see page 345), or proceed to Day 2 Operations in the link below.

4.11.3.6.4 Next Step:

AWS FIPS: Subsequent New Clusters (see page 345)

4.11.3.7 AWS FIPS: Create a Managed Cluster Using the DKP CLI

Enterprise Gov Advanced

In previous steps, you created a new Management cluster, which is self-managed. When you use these steps to create new Managed clusters, they will become Attached clusters under your Management Cluster.

When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

4.11.3.7.1 Create New Managed FIPS Cluster with the CLI

The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.

If you do not set a workspace, it will be created in the `default` workspace, and you need to take additional steps to attach to a workspace later. For instructions on how to do this, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster). (see page 784).

Your new, managed cluster needs to be part of either a new Workspace or an existing Workspace under a management cluster. Create or locate a workspace name to get started with your managed cluster.

4.11.3.7.2 Make New Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠️ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.11.3.7.3 Name Your Cluster

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS, it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

   ```
   export MANAGED_CLUSTER_NAME=<aws-additional>
   ```

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[304] for more naming information.

3. Execute this command to create your Kubernetes cluster using any relevant flags but note that the `--self-managed` is absent so that this cluster can be attached to your previously created Management Cluster.

Execute this command:

```
dkp create cluster aws \
--cluster-name=${MANAGED_CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
```

304 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
--namespace ${WORKSPACE_NAMESPACE}
--with-aws-bootstrap-credentials=true \
--kubernetes-version=v1.28.7+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere
--kubeconfig=<management-cluster-kubeconfig-path>
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

4.11.3.7.4 Retrieve the kubeconfig and Explore New AWS Cluster

Follow these steps:

1. Fetch the kubeconfig file with the command:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} --kubeconfig
   <management-cluster-kubeconfig-path> -n ${WORKSPACE_NAMESPACE} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

2. List the nodes with the following command:

   ```
   kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get nodes
   ```

3. List the pods with the following command:

NOTE: Wait for the Status to move to Ready while `calico-node` pods are being deployed.

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get pods -A
```

4.11.3.7.4.1 Manually Attach a DKP CLI Cluster to the Management Cluster

> ⚠ These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

   ```
   echo ${MANAGED_CLUSTER_NAME}
   ```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
   NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

   ```
   kubectl get workspaces -A
   ```

5. Set the WORKSPACE_NAMESPACE environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace-namespace>
   ```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

   ```
   kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
   template='{{.data.value}}{{ "\n"}}'
   ```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

After you have existing clusters or want to create new clusters to attach, there are many ways to attach a cluster with various requirements and restrictions. To see all the options, visit the section in documentation Day 2 - Attach an Existing Kubernetes Cluster (see page 784).

4.11.3.7.4.2 Next Step:

Day 2 - Cluster Operations Management (see page 590)

4.11.4 AWS Air-gapped FIPS Install

This section provides instructions to install DKP in an AWS air-gapped FIPS environment.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

4.11.4.1 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create permissions and roles on the Minimal Permissions and Role to Create Clusters (see page 1156) page.
2. Create Cluster IAM Policies and Roles (see page 1162).
3. Export the AWS region where you want to deploy the cluster:

   ```
   export AWS_REGION=us-west-2
   ```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

   ```
   export AWS_PROFILE=<profile>
   ```

If using AWS ECR as your local private registry, more information can be found on the Registry Mirror Tools (see page 1606) page.

To deploy a cluster with a custom image in a region where CAPI images[305] are not provided, you need to use Konvoy Image Builder (see page 1153) to create your own image for the region.

For multi-tenancy, every tenant should be in a different AWS account to ensure they are truly independent of other tenants in order to enforce security.

305 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html

For air-gapped, ensure you have downloaded (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , so you can extract the tarball on the page with those instructions.

4.11.4.2 Next Step:

AWS Air-gapped FIPS: Create an Image (see page 351)

4.11.4.3 AWS Air-gapped FIPS: Create an Image

To create an Image using Konvoy Image Builder (KIB) for use in an air-gapped cluster, follow these instructions. AMI images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create an AMI image of your current computer system settings and software. The AMI image can then be replicated and distributed, creating your computer system for other users. You can use overrides files to customize some of the components installed on your machine image. For example, you could tell KIB to install the FIPS versions of the Kubernetes components.

4.11.4.3.1 Prerequisites

- Minimal IAM Permissions for KIB (see page 1629) to create an Image for an AWS account using Konvoy Image Builder.

In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1626).
Explore the Customize your Image (see page 1661) topic for more options.
Using KIB (see page 1626), you can build an AMI without requiring access to the internet by providing an additional `--override` flag.

In previous DKP releases, the distro package bundles were included in the downloaded air-gapped bundle. Currently, that air-gapped bundle contains the following artifacts with the exception of the distro packages:

- DKP Kubernetes packages
- Python packages (provided by upstream)
- Containerd tarball

1. Download (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz && cd dkp-v2.8.1/kib
```

2. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.
3. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle`. This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml`. Example command:

   ```
   ./konvoy-image create-package-bundle --os redhat-8.4 --output-
   directory=artifacts
   ```

   NOTE: For FIPS, pass the flag: `--fips`

   NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY`. Example command:

   ```
   export RHSM_ACTIVATION_KEY="-ci"
   export RHSM_ORG_ID="1232131"
   ```

4. Follow the instructions below to build an AMI.

4.11.4.3.1.1 Offline FIPS Override File (Air-gapped)

Add the following FIPS offline override file to your environment:

`--overrides overrides/offline-fips.yaml`

```
# fips os-packages
os_packages_local_bundle_file: "{{ playbook_dir }}/../artifacts/
{{ kubernetes_version }}_{{ ansible_distribution|lower }}
_{{ ansible_distribution_major_version }}_x86_64_fips.tar.gz"
containerd_local_bundle_file: "{{ playbook_dir }}/../artifacts/
{{ containerd_tar_file }}"
pip_packages_local_bundle_file: "{{ playbook_dir }}/../artifacts/pip-packages.tar.gz"
images_local_bundle_dir: "{{ playbook_dir}}/../artifacts/images"
```

You can find all available Overrides files in the Konvoy Image Builder repo[306].

306 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

- An air-gapped example of override file use is the command below which produces an AWS FIPS-compliant image on RHEL 8.4:

```
konvoy-image build --overrides overrides/offline-fips.yaml images/ami/rhel-84.yaml
```

4.11.4.3.1.2 Next Step:

AWS Air-gapped FIPS: Load the Registry (see page 353)

4.11.4.4 AWS Air-gapped FIPS: Load the Registry

After you create an image for your air-gapped environment, you will need to load it into your registry.

4.11.4.4.1 Load Images into your Registry

Before creating an air-gapped Kubernetes cluster, you need to load the required images in a local registry for the Konvoy component. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1606) page for more information.

1. If not already done in prerequisites, download (see page 76) the air-gapped bundle `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap cluster, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

   ```
   cd dkp-v2.8.1
   ```

3. Set an environment variable with your registry address and any other needed variables using this command:

   ```
   export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
   export REGISTRY_USERNAME=<username>
   ```

```
export REGISTRY_PASSWORD=<password>
export REGISTRY_CA=<path to the cacert file on the bastion>
```

4. Execute the following command to load the air-gapped image bundle into your private registry using any of the relevant flags to apply variables above:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --
to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-
registry-password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

4.11.4.4.2 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

4.11.4.4.3 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

4.11.4.4.4 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME}
--to-registry-password=${REGISTRY_PASSWORD}
```

4.11.4.4.5 Next Step:

AWS Air-gapped FIPS: Create the Management Cluster (see page 355)

4.11.4.5 AWS Air-gapped FIPS: Create the Management Cluster

Create a new self-managed AWS Kubernetes cluster in an Air-gapped FIPS environment on your AWS infrastructure. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

To increase Docker Hub's rate limit[307] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster` command.

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable to the name you assigned this cluster:

   ```
   export CLUSTER_NAME=<aws-example>
   ```

   NOTE: The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[308] for more naming information.

3. Export variables for the existing infrastructure details:

   ```
   export AWS_VPC_ID=<vpc-...>
   export AWS_SUBNET_IDS=<subnet-...,subnet-...,subnet-...>
   export AWS_ADDITIONAL_SECURITY_GROUPS=<sg-...>
   export AWS_AMI_ID=<ami-...>
   ```

307 https://docs.docker.com/docker-hub/download-rate-limit/
308 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

- `AWS_VPC_ID` : the VPC ID where the cluster will be created. The VPC requires the following AWS VPC Endpoints to be already present:
  - `ec2` - `com.amazonaws.{region}.ec2`
  - `elasticloadbalancing` - `com.amazonaws.{region}.elasticloadbalancing`
  - `secretsmanager` - `com.amazonaws.{region}.secretsmanager`
  - `autoscaling` - `com.amazonaws.{region}.autoscaling`
  - `ecr` - `com.amazonaws.{region}.ecr.api` - `(authentication)`
  - `ecr` - `com.amazonaws.{region}.ecr.dkr` - `(data trasfer)`

  More details about AWS service using an interface VPC endpoint[309] and AWS VPC endpoints list[310].
- `AWS_SUBNET_IDS` : a comma-separated list of one or more private Subnet IDs with each one in a different Availability Zone. The cluster control-plane and worker nodes will automatically be spread across these Subnets.
- `AWS_ADDITIONAL_SECURITY_GROUPS` : a comma-seperated list of one or more Security Groups IDs to use in addition to the ones automatically created by CAPA[311].
- `AWS_AMI_ID` : the AMI ID to use for control-plane and worker nodes. The AMI must be created by the konvoy-image-builder (see page 1626).

> ⚠ In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1626).

There are two approaches to supplying the ID of your AMI while creating your cluster.

- **Option One** is to provide the ID of the AMI: `--ami AMI_ID` .
- **Option Two** is provide a way for DKP to discover the AMI using location, format and OS information using flags: AWS Account ID: `--ami-owner AWS_ACCOUNT_ID` , format `--ami-base-os ubuntu-20.04` , and OS `--ami-format 'example-{{.BaseOS}}-?{{.K8sVersion}}-*'` . Examples of these choices are shown in the `dkp create cluster aws` code snippets below.

309 https://docs.aws.amazon.com/vpc/latest/privatelink/create-interface-endpoint.html
310 https://docs.aws.amazon.com/vpc/latest/privatelink/aws-services-privatelink-support.html
311 https://github.com/kubernetes-sigs/cluster-api-provider-aws

4. ⚠ **IMPORTANT:** You must tag the subnets as described below to allow for Kubernetes to create External Load Balancers (ELBs) for services of type `LoadBalancer` in those subnets. If the subnets are not tagged, they will not receive an ELB and the following error displays: `Error syncing load balancer, failed to ensure load balancer; could not find any suitable subnets for creating the ELB.`.

   The tags should be set as follows, where `<CLUSTER_NAME>` corresponds to the name set in `CLUSTER_NAME` environment variable:

   ```
   kubernetes.io/cluster = <CLUSTER_NAME>
   kubernetes.io/cluster/<CLUSTER_NAME> = owned
   kubernetes.io/role/internal-elb = 1
   ```

5. (Optional) Configure your cluster to use an existing container registry as a mirror when attempting to pull images. The example below is for AWS ECR:
   ⚠ If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1606) page for more information.

   ⚠ **IMPORTANT:** The AMI must be created by the konvoy-image-builder (see page 1626) project in order to use the registry mirror feature.

   ```
   export REGISTRY_URL=<ecr-registry-URI>
   ```

   - `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
   - NOTE: Other local registries may use the options below:
     - JFrog - `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
     - `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
     - `REGISTRY_PASSWORD` : optional if username is not set.

6. Create a Kubernetes cluster. The following example shows a common configuration. See dkp create cluster aws (see page 1856) reference for the full list of cluster creation options:

⚠ DKP uses local static provisioner as the default storage provider (see page 110). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[312] compatible storage that is suitable for production.

312 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

- You can choose from any of the storage options[313] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[314].

Run this command to create your Kubernetes cluster using any relevant flags for **Option One** explained above providing the AMI ID:

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--with-aws-bootstrap-credentials=true \
--vpc-id=${AWS_VPC_ID} \
--ami=${AWS_AMI_ID} \
--subnet-ids=${AWS_SUBNET_IDS} \
--internal-load-balancer=true \
--additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
--registry-mirror-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD} \
--kubernetes-version=v1.28.7+fips.0 \
--etcd-version=3.5.10+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
--self-managed
```

OR

Option Two is to run the command as shown from the explanation above providing the location, format and base OS of your image:

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--with-aws-bootstrap-credentials=true \
--vpc-id=${AWS_VPC_ID} \
--ami=${AWS_AMI_ID} \
--ami-owner AWS_ACCOUNT_ID \
--ami-base-os ubuntu-20.04 \
--ami-format 'example-{{.BaseOS}}-?{{.K8sVersion}}-*' \
--subnet-ids=${AWS_SUBNET_IDS} \
--internal-load-balancer=true \
--additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
--registry-mirror-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD} \
--kubernetes-version=v1.28.7+fips.0 \
--etcd-version=3.5.10+fips.0 \
```

313 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
314 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

```
--kubernetes-image-repository=docker.io/mesosphere \
--self-managed
```

Flatcar OS

Flatcar OS (see page 1183) use `--os-hint` to instruct the bootstrap cluster to make some changes related to the installation paths:

```
--os-hint flatcar
```

> If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

You can find a customizable Create a New AWS Cluster[315] under the Custom Installation and Additional Infrastructure Tools (see page 1050). For more details about FIPS, refer to the FIPS 140-2 Compliance (see page 1598) section of the documentation.

4.11.4.5.1 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.11.4.5.2 Next Step:

AWS Air-gapped FIPS: Install Kommander (see page 359)

4.11.4.6 AWS Air-gapped FIPS: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

4.11.4.6.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry: Air-gapped Environments (see page 1536).

315 https://d2iq.atlassian.net/wiki/pages/createpage.action?fromPageId=161448833&linkCreation=true&spaceKey=DENT&title=AWS+Cluster+Creation+Choices

- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

4.11.4.6.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init --airgapped > kommander.yaml
   ```

4. *If required*, **customize** your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include:
   - Custom Domains and Certificates
   - HTTP proxy
   - External Load Balancer
   - GPU utilization, etc.
   - Rook Ceph customization for Pre-provisioned environments (see page 1541)

5. *If required*: If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

   ```
   ...
   apps:
     traefik:
       enabled: true
       values: |
         service:
           annotations:
             service.beta.kubernetes.io/aws-load-balancer-internal: "true
   ...
   ```

6. Expand **one** of the following sets of instructions, depending on your license and application environments:

4.11.4.6.1.2 Enable DKP Catalog Applications and Install Kommander in an Air-gapped Environment

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add the following values to enable DKP Catalog Applications:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./dkp-catalog-applications-v2.8.1.tar.gz
```

⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.1.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.1.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.8.1.tar.gz
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).

- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.11.4.6.1.3 Next Step:

AWS Air-gapped FIPS: Verify Install and Log in the UI (see page 362)

4.11.4.7 AWS Air-gapped FIPS: Verify Install and Log in the UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.11.4.7.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

4.11.4.7.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

4.11.4.7.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

Now you can create additional new clusters (see page 364), or proceed to Day 2 Operations in the link below.

4.11.4.7.4 Next Step:

AWS Air-gapped FIPS: Subsequent New Clusters (see page 364)

4.11.4.8 AWS Air-gapped FIPS: Create a Managed Cluster Using the DKP CLI

Enterprise Gov Advanced

In previous steps, you created a new Management cluster, which is self-managed. When you use these steps to create new Managed clusters, they will become Attached clusters under your Management Cluster.

⚠ When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

4.11.4.8.1 Create New Managed Air-gapped FIPS Cluster with the CLI

The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.
If you do not set a workspace, it will be created in the `default` workspace, and you need to take additional steps to attach to a workspace later. For instructions on how to do this, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster). (see page 784).

Your new, managed cluster needs to be part of either a new Workspace or an existing Workspace under a management cluster. Create or locate a workspace name to get started with your managed cluster.

4.11.4.8.2 Make New Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.11.4.8.3 Name Your Cluster

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

```
export MANAGED_CLUSTER_NAME=<aws-additional>
```

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[316] for more naming information.

3. Execute this command to create your Kubernetes cluster using any relevant flags but note that the `--self-managed` is absent so that this cluster can be attached to your previously created Management Cluster:

```
dkp create cluster aws
--cluster-name=${MANAGED_CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--namespace ${WORKSPACE_NAMESPACE}
--with-aws-bootstrap-credentials=true \
--vpc-id=${AWS_VPC_ID} \
--ami=${AWS_AMI_ID} \
--subnet-ids=${AWS_SUBNET_IDS} \
--internal-load-balancer=true \
--additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
--registry-mirror-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD} \
--kubeconfig=<management-cluster-kubeconfig-path> \
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

4.11.4.8.4 Retrieve the kubeconfig and Explore New AWS Cluster

Follow these steps:

1. Fetch the kubeconfig file with the command:

316 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} --kubeconfig
<management-cluster-kubeconfig-path> -n ${WORKSPACE_NAMESPACE} > $
{MANAGED_CLUSTER_NAME}.conf
```

2. List the nodes with the following command:

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get nodes
```

3. List the pods with the following command:

NOTE: Wait for the Status to move to Ready while `calico-node` pods are being deployed.

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get pods -A
```

4.11.4.8.4.1 Manually Attach a DKP CLI Cluster to the Management Cluster

These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

```
echo ${MANAGED_CLUSTER_NAME}
```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
{MANAGED_CLUSTER_NAME}.conf
```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
   NOTE: This is only necessary if you never set the workspace of your cluster upon creation.
4. Retrieve the workspace where you want to attach the cluster:

```
kubectl get workspaces -A
```

5. Set the WORKSPACE_NAMESPACE environment variable:

```
export WORKSPACE_NAMESPACE=<workspace-namespace>
```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

```
kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
template='{{.data.value}}{{ "\n"}}'
```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
```

```
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

When you have existing clusters or want to create other new clusters to attach, there are many ways to attach a cluster with various requirements and restrictions. To see all the options, visit the section in the Day 2- Cluster Operations Management section for documentation on how to Attach an Existing Kubernetes Cluster (see page 784).

4.11.4.8.4.2 Next Step:

Day 2 - Cluster Operations Management (see page 590)

4.11.5 AWS with GPU Install

This section provides instructions to install DKP in an AWS non-air-gapped GPU environment.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

4.11.5.1 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create permissions and roles on the Minimal Permissions and Role to Create Clusters (see page 1156) page.
2. Create Cluster IAM Policies and Roles (see page 1162).
3. Export the AWS region where you want to deploy the cluster:

```
export AWS_REGION=us-west-2
```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

```
export AWS_PROFILE=<profile>
```

If using AWS ECR as your local private registry, more information can be found on the Registry Mirror Tools (see page 1606) page.

To deploy a cluster with a custom image in a region where CAPI images[317] are not provided, you need to use Konvoy Image Builder (see page 1153) to create your own image for the region.

For multi-tenancy, every tenant should be in a different AWS account to ensure they are truly independent of other tenants in order to enforce security.

4.11.5.2 GPU Prerequisites

Before you begin, you must:

- Ensure nodes provide an NVIDIA GPU.
- If you are using a public cloud service such as AWS (see page 1249), create an AMI with KIB using the instructions on the KIB for GPU (see page 1649) page.
- If you are deploying in a pre-provisioned environment, ensure that you have created the appropriate secret for your GPU nodepool (see page 239) and have uploaded the appropriate artifacts to each node. See the GPU only steps section (see page 0)on the Pre-provisioned Prerequisites Air-gapped page for additional information.

Specific instructions must be followed for enabling `nvidia-gpu-operator` depending on if you want to deploy the app on a Management cluster or a Attached or a Managed cluster.

- For instructions on enabling the NVIDIA platform application on a Management cluster, follow the instructions in the NVIDIA Platform Application Management Cluster (see page 1000) section.

317 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html

- For instructions on enabling the NVIDIA platform application on attached or managed clusters, follow the instructions in the NVIDIA Platform Application Attached or Managed Cluster (see page 1002) section.

After `nvidia-gpu-operator` has been enabled depending on the cluster type, proceed to the **Select the correct Toolkit version for your NVIDIA GPU Operator** on each of those pages.

4.11.5.3 Next Step:

AWS GPU: Use Node Label Automatic Configuration (see page 371)

4.11.5.4 AWS GPU: Use Node Label Automatic Configuration

When using GPU nodes, it is important they have the proper label identifying them as Nvidia GPU nodes. Node feature discovery (NFD), by default labels PCI hardware as:

```
"feature.node.kubernetes.io/pci-<device label>.present": "true"
```

where `<device label>` is by default as defined in this topic[318]:

```
< class > _ < vendor >
```

However, because there is a wide variety in devices and their assigned PCI classes, you may find that the labels assigned to your GPU nodes do not always properly identify them as containing an Nvidia GPU.

If the default detection does not work, you can manually change the daemonset that the GPU operator creates by running the following command:

```
        nodeSelector:
          feature.node.kubernetes.io/pci-< class > _ < vendor>.present: "true"
```

where `class` is any 4 digit number starting with `03xy` and the vendor for Nvidia is `10de`. If this is already deployed, you can always change the `daemonset` and change the `nodeSelector` field so that it deploys to the right nodes.

4.11.5.4.1 Next Step:

AWS GPU: Create an Image (see page 372)

318 https://kubernetes-sigs.github.io/node-feature-discovery/v0.7/get-started/features.html#pci

4.11.5.5 AWS GPU: Create an Image

4.11.5.5.1 Learn how to build a custom AMI for use with DKP

AMI images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create an AMI image of your current computer system settings and software. The AMI image can then be replicated and distributed, creating your computer system for other users. You can use overrides files to customize some of the components installed on your machine image. For example, you could tell KIB to install the FIPS versions of the Kubernetes components.

This procedure describes how to use the Konvoy Image Builder (see page 1626) (KIB) to create a Cluster API[319] compliant Amazon Machine Image (AMI). KIB uses variable overrides (see page 1670) to specify base image and container images to use in your new AMI.

> ⚠ In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1629). Explore the Customize your Image (see page 1661) topic for more options about overrides.

4.11.5.5.2 Prerequisites

Before you begin, you must:

- Download the KIB (see page 0) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Check the Supported Infrastructure Operating Systems (see page 67)
- Check the Supported Kubernetes Version (see page 1706) for your Provider.
- Create a working registry:
  - Version 4.0 of Podman[320]or higher for Linux. Host requirements found here: https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
  - Version 20.10.0 of Docker[321] or higher for Linux or MacOS
- Ensure you have met the minimal set of permissions from the AWS Image Builder Book[322].
- A Minimal IAM Permissions for KIB (see page 1629) to create an Image for an AWS account using Konvoy Image Builder.

319 https://cluster-api.sigs.k8s.io/
320 https://podman.io/getting-started/installation
321 https://www.docker.com/
322 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis

4.11.5.5.2.1 Extract KIB Bundle

If not done previously during Konvoy Image Builder download in Prerequisites, extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION` folder. **Otherwise, proceed to Build the Image.**

In previous DKP releases, the distro package bundles were included in the downloaded air-gapped bundle. Currently, that air-gapped bundle contains the following artifacts with the exception of the distro packages:

- DKP Kubernetes packages
- Python packages (provided by upstream)
- Containerd tarball

1. Download (see page 76) `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz && cd dkp-v2.8.0/kib
   ```

2. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.
3. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle` . This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml` . Example command:

   ```
   ./konvoy-image create-package-bundle --os redhat-8.4 --output-
   directory=artifacts
   ```

   NOTE: For FIPS, pass the flag: `--fips`
   NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY` . Example command:

   ```
   export RHSM_ACTIVATION_KEY="-ci"
   export RHSM_ORG_ID="1232131"
   ```

4. Follow the instructions below to build an AMI.

The `konvoy-image` binary and all supporting folders are also extracted. When run, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

- Set environment variables for AWS access[323]. The following variables must be set using your credentials including required IAM (see page 1629):

```
export AWS_ACCESS_KEY_ID
export AWS_SECRET_ACCESS_KEY
export AWS_DEFAULT_REGION
```

- If you have an override file (see page 1670) to configure specific attributes of your AMI file, add it. Instructions for customizing an override file are found on this page: Image Overrides (see page 1678)

4.11.5.5.2.2 Build the GPU Image

Using the Konvoy Image Builder (see page 1626), you can build an image that has support to use NVIDIA GPU hardware to support GPU workloads.

> **NOTE:** The NVIDIA driver requires a specific Linux kernel version. Make sure that the base image for the OS version has the required kernel version.
>
> See Supported Infrastructure Operating Systems (see page 67) for a list of OS versions and the corresponding kernel versions known to work with the NVIDIA driver.

If the NVIDIA runfile[324] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory (you must create an `artifacts` directory if you don't already have one).

```
curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/NVIDIA-Linux-
x86_64-470.82.01.run
mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
```

> DKP supported NVIDIA driver[325] version is 470.x.

To build an image for use on GPU enabled hardware, perform the following steps.

323 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-envvars.html
324 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile
325 https://www.nvidia.com/Download/Find.aspx

1. In your override file, add the following to enable GPU builds. Otherwise, you can access and use the overrides in our repo[326] or in the documentation under Nvidia GPU Override File (see page 1675) or Offline Nvidia Override file (see page 1676).
   a. Non-air-gapped GPU override `overrides/nvidia.yaml`:

```
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
```

   b. Air-gapped GPU override `overrides/offline-nvidia.yaml`:

```
# Use this file when building a machine image, not as a override secret
for preprovisioned environments
nvidia_runfile_local_file: "{{ playbook_dir}}/../artifacts/
{{ nvidia_runfile_installer }}"
gpu:
  types:
    - nvidia

build_name_extra: "-nvidia"
```

   NOTE: For RHEL Pre-provisioned Override Files (see page 1680) used with KIB, see specific note for GPU.

2. Build your image using the `build` Konvoy Image Builder command, making sure to include the flag `--instance-type` that specifies an AWS instance that has an available GPU:
   AWS Example:

```
konvoy-image build --region us-east-1 --instance-type=p2.xlarge --source-
ami=ami-12345abcdef images/ami/centos-7.yaml --overrides overrides/nvidia.yaml
```

   In this example, we chose an instance type with an NVIDIA GPU using the `--instance-type` flag, and we provided the NVIDIA overrides using the `--overrides` flag. See Using KIB with AWS (see page 1629) for more information on creating an AMI.

3. When the Konvoy Image Builder image is complete, the custom `ami` id is printed and written to `./.`. To use the built `ami` with Konvoy, specify it with the `--ami` flag when calling cluster create.

Additional helpful information can be found in the NVIDIA Device Plug-in[327] for Kubernetes instructions and the Installation Guide of Supported Platforms[328].

326 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides
327 https://github.com/NVIDIA/k8s-device-plugin/blob/master/README.md
328 https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html

See also: NVIDIA documentation[329]

Related Information:

- To use a local registry even in a non-air-gapped environment, download and extract the bundle. Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.7.0_linux_amd64.tar.gz`) to load registry.
- Load the Registry (see page 317)

4.11.5.5.2.3 Next Steps:

AWS GPU: Create the Management Cluster (see page 376)

4.11.5.6 AWS GPU: Create the Management Cluster

Use this procedure to create a self-managed AWS GPU Management cluster with DKP. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing. After the GPU compatible image is created with KIB, a cluster can be generated using that custom AMI.

4.11.5.6.1 Name Your Cluster

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[330] for more naming information.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

```
export CLUSTER_NAME=<aws-example>
```

329 https://nvidia.custhelp.com/app/answers/detail/a_id/131/kw/driver%20installation%20docs/related/1
330 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

4.11.5.6.2 Create a New AWS Kubernetes Cluster

If you use these instructions to create a cluster on AWS using the DKP default settings without any edits to configuration files or additional flags, your cluster is deployed on an Ubuntu 20.04 operating system image (see page 67) with 3 control plane nodes, and 4 worker nodes.

DKP uses AWS CSI as the default storage provider (see page 110). You can use a Kubernetes CSI[331] compatible storage solution that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[332] for more information.

> ⚠ In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1626).

There are two approaches to supplying the ID of your AMI. Either provide the ID of the AMI or provide a way for DKP to discover the AMI using location, format and OS information:

1. **Option One** - Provide the ID of your AMI:
   a. Use the example command below leaving the existing flag that provides the AMI ID: `--ami AMI_ID`
2. **Option Two** - Provide a path for your AMI with the information required for image discover:
   a. Where the AMI is published using your AWS Account ID: `--ami-owner AWS_ACCOUNT_ID`
   b. The format or string used to search for matching AMIs and ensure it references the Kubernetes version plus the base OS name: `--ami-base-os ubuntu-20.04`
   c. The base OS information: `--ami-format 'example-{{.BaseOS}}-?{{.K8sVersion}}-*'`

- ⚠ **IMPORTANT:** The AMI must be created with Konvoy Image Builder (see page 1626) in order to use the registry mirror feature.

```
export AWS_AMI_ID=<ami-...>
```

- **(Optional) Registry Mirror** - Configure your cluster to use an existing local registry as a mirror when attempting to pull images. Below is an AWS ECR example:

```
export REGISTRY_URL=<ecr-registry-URI>
```

331 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
332 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

- `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.

1. Execute this command to create a cluster with a GPU AMI and `--self-managed` flag. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing:

```
dkp create cluster aws \
--cluster-name=${CLUSTER_NAME} \
--with-aws-bootstrap-credentials=true \
--self-managed
```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy` , `--https-proxy` , and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

Flatcar OS

Flatcar OS (see page 1183) use `--os-hint` to instruct the bootstrap cluster to make some changes related to the installation paths:

```
--os-hint flatcar
```

2. Create the node pool after cluster creation:

```
dkp create nodepool aws -c ${CLUSTER_NAME} \
--instance-type p2.xlarge \
--ami-id=${AMI_ID_FROM_KIB} \
--replicas=1 ${NODEPOOL_NAME} \
--kubeconfig=${CLUSTER_NAME}.conf
```

You can find a customizable Create a New AWS Cluster[333] under Custom Installation and Additional Infrastructure Tools (see page 1050) .

4.11.5.6.3 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

333 https://d2iq.atlassian.net/wiki/pages/createpage.action?fromPageId=162038621&linkCreation=true&spaceKey=DENT&title=AWS+Cluster+Creation+Choices

4.11.5.6.4 Next Step:

AWS GPU: Install Kommander (see page 379)

4.11.5.7 AWS GPU: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

4.11.5.7.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

4.11.5.7.1.1 Create and customize your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init > kommander.yaml
   ```

4. *If required:* Customize your `kommander.yaml` .
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1541)), etc.

5. *If required:* If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

   ```
   apps:
   ```

```
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true"
```

4.11.5.7.1.2 Enable GPU Resources

1. In the same `kommander.yaml` file, enable Nvidia platform services.

```
apps:
  nvidia-gpu-operator:
   enabled: true
```

2. Append the correct Toolkit version based on your OS:

   The **NVIDIA Container Toolkit** allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

   Centos 7.9/RHEL 7.9:
   If you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.14.6-centos7
```

   RHEL 8.4/8.6 and SLES 15 SP3
   If you're using RHEL 8.4/8.6 or SLES 15 SP3 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
```

```
      toolkit:
        version: v1.14.6-ubi8
```

Ubuntu 18.04 and 20.04
If you're using Ubuntu 18.04 or 20.04 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.14.6-ubuntu20.04
```

4.11.5.7.1.3 Enable DKP Catalog Applications and Install Kommander

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.7.0
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.11.5.7.1.4 Next Step:

AWS GPU: Verify Install and Log in the UI (see page 382)

4.11.5.8 AWS GPU: Verify Install and Log in the UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.11.5.8.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

4.11.5.8.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

4.11.5.8.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

For more GPU information (see page 998), see the section for GPU under Day 2 - Cluster Operations Management.

4.11.5.8.4 Next Step:

AWS GPU: Subsequent New Clusters (see page 385)

4.11.5.9 AWS GPU: Create a Managed Cluster Using the DKP CLI

Enterprise Gov Advanced

In previous steps, you created a new Management cluster, which is self-managed. When you use these steps to create new Managed clusters, they will become Attached clusters under your Management Cluster.

When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

4.11.5.9.1 Create New Managed GPU Cluster with the CLI

The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.
If you do not set a workspace, it will be created in the `default` workspace, and you need to take additional steps to attach to a workspace later. For instructions on how to do this, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster). (see page 784).

Your new, managed cluster needs to be part of either a new Workspace or an existing Workspace under a management cluster. Create or locate a workspace name to get started with your managed cluster.

4.11.5.9.2 Make New Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.11.5.9.3 Name Your Cluster

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

```
export MANAGED_CLUSTER_NAME=<aws-additional>
```

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[334] for more naming information.

3. Execute this command to create your Kubernetes cluster using any relevant flags but note that the `--self-managed` is absent so that this cluster can be attached to your previously created Management Cluster.

- Execute this command to create a cluster with a GPU AMI:

```
dkp create cluster aws \
--cluster-name=${MANAGED_CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--kubeconfig=<management-cluster-kubeconfig-path> \
--namespace ${WORKSPACE_NAMESPACE}
--with-aws-bootstrap-credentials=true \
--kubeconfig=<management-cluster-kubeconfig-path> \
```

  If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

- Create the node pool after cluster creation:

```
dkp create nodepool aws -c ${CLUSTER_NAME} \
--instance-type p2.xlarge \
--ami-id=${AMI_ID_FROM_KIB} \
--replicas=1 ${NODEPOOL_NAME} \
--kubeconfig=${CLUSTER_NAME}.conf \
```

334 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

4.11.5.9.4 Retrieve the kubeconfig and Explore New AWS Cluster

Follow these steps:

1. Fetch the kubeconfig file with the command:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} --kubeconfig
<management-cluster-kubeconfig-path> -n ${WORKSPACE_NAMESPACE} > $
{MANAGED_CLUSTER_NAME}.conf
```

2. List the nodes with the following command:

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get nodes
```

3. List the pods with the following command:

NOTE: Wait for the Status to move to Ready while `calico-node` pods are being deployed.

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get pods -A
```

4.11.5.9.4.1 Manually Attach a DKP CLI Cluster to the Management Cluster

These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

```
echo ${MANAGED_CLUSTER_NAME}
```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
{MANAGED_CLUSTER_NAME}.conf
```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

```
kubectl get workspaces -A
```

5. Set the WORKSPACE_NAMESPACE environment variable:

```
export WORKSPACE_NAMESPACE=<workspace-namespace>
```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

```
kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
template='{{.data.value}}{{ "\n"}}'
```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

4.11.5.9.4.2 Next Step:

Day 2 - Cluster Operations Management (see page 590)

4.11.6 AWS Air-gapped with GPU Install

This section provides instructions to install DKP in an AWS air-gapped GPU environment.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

> For air-gapped, ensure you have downloaded (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`, so you can extract the tarball on the page with those instructions.

4.11.6.1 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create a Minimal Permissions and Role to Create Clusters (see page 1156)..
2. Create Cluster IAM Policies and Roles (see page 1162).
3. Export the AWS region where you want to deploy the cluster:

   ```
   export AWS_REGION=us-west-2
   ```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

   ```
   export AWS_PROFILE=<profile>
   ```

4.11.6.1.1 GPU Prerequisites

Before you begin, you must:

- Ensure nodes provide an NVIDIA GPU.
- If you are using a public cloud service such as AWS (see page 1249), create an AMI with KIB using the instructions on the KIB for GPU (see page 1649) page.
- If you are deploying in a pre-provisioned environment, ensure that you have created the appropriate secret for your GPU nodepool (see page 239) and have uploaded the appropriate artifacts to each node. See the GPU only steps section (see page 0)on the Pre-provisioned Prerequisites Air-gapped page for additional information.

Specific instructions must be followed for enabling `nvidia-gpu-operator` depending on if you want to deploy the app on a Management cluster or a Attached or a Managed cluster.

- For instructions on enabling the NVIDIA platform application on a Management cluster, follow the instructions in the NVIDIA Platform Application Management Cluster (see page 1000) section.
- For instructions on enabling the NVIDIA platform application on attached or managed clusters, follow the instructions in the NVIDIA Platform Application Attached or Managed Cluster (see page 1002) section.

After `nvidia-gpu-operator` has been enabled depending on the cluster type, proceed to the **Select the correct Toolkit version for your NVIDIA GPU Operator** on each of those pages.

4.11.6.1.2 Next Step:

AWS Air-gapped GPU: Use Node Label Automatic Configuration (see page 391)

4.11.6.2 AWS Air-gapped GPU: Use Node Label Automatic Configuration

When using GPU nodes, it is important they have the proper label identifying them as Nvidia GPU nodes. Node feature discovery (NFD), by default labels PCI hardware as:

```
"feature.node.kubernetes.io/pci-<device label>.present": "true"
```

where `<device label>` is by default as defined in this topic[335]:

```
< class > _ < vendor >
```

However, because there is a wide variety in devices and their assigned PCI classes, you may find that the labels assigned to your GPU nodes do not always properly identify them as containing an Nvidia GPU.

If the default detection does not work, you can manually change the daemonset that the GPU operator creates by running the following command:

```
          nodeSelector:
            feature.node.kubernetes.io/pci-< class > _ < vendor>.present: "true"
```

where `class` is any 4 digit number starting with `03xy` and the vendor for Nvidia is `10de`. If this is already deployed, you can always change the `daemonset` and change the `nodeSelector` field so that it deploys to the right nodes.

4.11.6.2.1 Next Step:

AWS Air-gapped GPU: Create an Image (see page 391)

4.11.6.3 AWS Air-gapped GPU: Create an Image

To create an Image using Konvoy Image Builder (KIB) for use in an air-gapped cluster, follow these instructions. AMI images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create an AMI image of your current computer system settings and software. The AMI image can then be replicated and distributed, creating your computer system for other users. You can use overrides files to customize some of the components installed on your machine image. For example, you could tell KIB to install the FIPS versions of the Kubernetes components.

335 https://kubernetes-sigs.github.io/node-feature-discovery/v0.7/get-started/features.html#pci

4.11.6.3.1 Prerequisites

- Minimal IAM Permissions for KIB (see page 1629) to create an Image for an AWS account using Konvoy Image Builder.

> ⚠ In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1626).
> Explore the Customize your Image (see page 1661) topic for more options.
> Using KIB (see page 1626), you can build an AMI without requiring access to the internet by providing an additional `--override` flag.

In previous DKP releases, the distro package bundles were included in the downloaded air-gapped bundle. Currently, that air-gapped bundle contains the following artifacts with the exception of the distro packages:

- DKP Kubernetes packages
- Python packages (provided by upstream)
- Containerd tarball

1. Download (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz && cd dkp-v2.8.1/kib
   ```

2. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.
3. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle` . This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml` . Example command:

   ```
   ./konvoy-image create-package-bundle --os redhat-8.4 --output-
   directory=artifacts
   ```

   NOTE: For FIPS, pass the flag: `--fips`

   NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY` . Example command:

```
export RHSM_ACTIVATION_KEY="-ci"
export RHSM_ORG_ID="1232131"
```

4. Follow the instructions below to build an AMI.

4.11.6.3.1.1 Build the GPU Image

Using the Konvoy Image Builder (see page 1626), you can build an image that has support to use NVIDIA GPU hardware to support GPU workloads.

> **NOTE:** The NVIDIA driver requires a specific Linux kernel version. Make sure that the base image for the OS version has the required kernel version.
>
> See Supported Infrastructure Operating Systems (see page 67) for a list of OS versions and the corresponding kernel versions known to work with the NVIDIA driver.

If the NVIDIA runfile[336] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory (you must create an `artifacts` directory if you don't already have one).

```
curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/NVIDIA-Linux-
x86_64-470.82.01.run
mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
```

> DKP supported NVIDIA driver[337] version is 470.x.

To build an image for use on GPU enabled hardware, perform the following steps.

1. In your override file, add the following to enable GPU builds. Otherwise, you can access and use the overrides in our repo[338] or in the documentation under Nvidia GPU Override File (see page 1675) or Offline Nvidia Override file (see page 1676).
    a. Non-air-gapped GPU override `overrides/nvidia.yaml`:

336 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile
337 https://www.nvidia.com/Download/Find.aspx
338 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

```
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
```

b. Air-gapped GPU override `overrides/offline-nvidia.yaml`:

```
# Use this file when building a machine image, not as a override secret
for preprovisioned environments
nvidia_runfile_local_file: "{{ playbook_dir}}/../artifacts/
{{ nvidia_runfile_installer }}"
gpu:
  types:
    - nvidia

build_name_extra: "-nvidia"
```

NOTE: For RHEL Pre-provisioned Override Files (see page 1680) used with KIB, see specific note for GPU.

2. Build your image using the `build` Konvoy Image Builder command, making sure to include the flag `--instance-type` that specifies an AWS instance that has an available GPU:
   AWS Example:

```
konvoy-image build --region us-east-1 --instance-type=p2.xlarge --source-
ami=ami-12345abcdef images/ami/centos-7.yaml --overrides overrides/nvidia.yaml
```

   In this example, we chose an instance type with an NVIDIA GPU using the `--instance-type` flag, and we provided the NVIDIA overrides using the `--overrides` flag. See Using KIB with AWS (see page 1629) for more information on creating an AMI.

3. When the Konvoy Image Builder image is complete, the custom `ami` id is printed and written to `./.`. To use the built `ami` with Konvoy, specify it with the `--ami` flag when calling cluster create.

Additional helpful information can be found in the NVIDIA Device Plug-in[339] for Kubernetes instructions and the Installation Guide of Supported Platforms[340].

See also: NVIDIA documentation[341]

4.11.6.3.1.2 Next Steps:

AWS Air-gapped GPU: Load the Registry (see page 395)

339 https://github.com/NVIDIA/k8s-device-plugin/blob/master/README.md
340 https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html
341 https://nvidia.custhelp.com/app/answers/detail/a_id/131/kw/driver%20installation%20docs/related/1

4.11.6.4 AWS Air-gapped GPU: Load the Registry

After you create an image for your air-gapped environment, you will need to load it into your registry.

4.11.6.4.1 Load Images into your Registry

Before creating an air-gapped Kubernetes cluster, you need to load the required images in a local registry for the Konvoy component. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1606) page for more information.

1. If not already done in prerequisites, download (see page 76) the air-gapped bundle `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap cluster, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

   ```
   cd dkp-v2.8.1
   ```

3. Set an environment variable with your registry address and any other needed variables using this command:

   ```
   export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   export REGISTRY_CA=<path to the cacert file on the bastion>
   ```

4. Execute the following command to load the air-gapped image bundle into your private registry using any of the relevant flags to apply variables above:

   ```
   dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --
   to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-
   registry-password=${REGISTRY_PASSWORD}
   ```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

4.11.6.4.2 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

4.11.6.4.3 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-password=${REGISTRY_PASSWORD}
```

4.11.6.4.4 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-password=${REGISTRY_PASSWORD}
```

4.11.6.4.5 Next Step:

AWS Air-gapped GPU: Create the Management Cluster (see page 396)

4.11.6.5 AWS Air-gapped GPU: Create the Management Cluster

Use this procedure to create a self-managed air-gapped AWS GPU Management cluster with DKP. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are

running on the same cluster they are managing. To create a cluster in an AWS Air-gapped environment using GPUs, execute the following:

To increase Docker Hub's rate limit[342], use your Docker Hub credentials when creating the cluster by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster` command.

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable to the name you assigned this cluster:

   ```
   export CLUSTER_NAME=<aws-example>
   ```

   NOTE: The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[343] for more naming information.

3. Export variables for the existing infrastructure details:

   ```
   export AWS_VPC_ID=<vpc-...>
   export AWS_SUBNET_IDS=<subnet-...,subnet-...,subnet-...>
   export AWS_ADDITIONAL_SECURITY_GROUPS=<sg-...>
   export AWS_AMI_ID=<ami-...>
   ```

   - `AWS_VPC_ID`: the VPC ID where the cluster will be created. The VPC requires the following AWS VPC Endpoints to be already present:
     - `ec2` - `com.amazonaws.{region}.ec2`
     - `elasticloadbalancing` - `com.amazonaws.{region}.elasticloadbalancing`
     - `secretsmanager` - `com.amazonaws.{region}.secretsmanager`
     - `autoscaling` - `com.amazonaws.{region}.autoscaling`
     - `ecr` - `com.amazonaws.{region}.ecr.api` - `(authentication)`

342 https://docs.docker.com/docker-hub/download-rate-limit/
343 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

    - `ecr` - `com.amazonaws.{region}.ecr.dkr` - `(data trasfer)`

  More details about AWS service using an interface VPC endpoint[344] and AWS VPC endpoints list[345].
- `AWS_SUBNET_IDS` : a comma-separated list of one or more private Subnet IDs with each one in a different Availability Zone. The cluster control-plane and worker nodes will automatically be spread across these Subnets.
- `AWS_ADDITIONAL_SECURITY_GROUPS` : a comma-seperated list of one or more Security Groups IDs to use in addition to the ones automatically created by CAPA[346].
- `AWS_AMI_ID` : the AMI ID to use for control-plane and worker nodes. The AMI must be created by the konvoy-image-builder (see page 1626).

> ⚠ In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1626).

There are two approaches to supplying the ID of your AMI while creating your cluster.

- **Option One** is to provide the ID of the AMI: `--ami AMI_ID` .
- **Option Two** is provide a way for DKP to discover the AMI using location, format and OS information using flags: AWS Account ID: `--ami-owner AWS_ACCOUNT_ID` , format `--ami-base-os ubuntu-20.04` , and OS `--ami-format 'example-{{.BaseOS}}-?{{.K8sVersion}}-*'` . Examples of these choices are shown in the `dkp create cluster aws` code snippets below.

4. ⚠ **IMPORTANT:** You must tag the subnets as described below to allow for Kubernetes to create External Load Balancers (ELBs) for services of type `LoadBalancer` in those subnets. If the subnets are not tagged, they will not receive an ELB and the following error displays: `Error syncing load balancer, failed to ensure load balancer; could not find any suitable subnets for creating the ELB.` .

   The tags should be set as follows, where `<CLUSTER_NAME>` corresponds to the name set in `CLUSTER_NAME` environment variable:

   ```
   kubernetes.io/cluster = <CLUSTER_NAME>
   kubernetes.io/cluster/<CLUSTER_NAME> = owned
   kubernetes.io/role/internal-elb = 1
   ```

344 https://docs.aws.amazon.com/vpc/latest/privatelink/create-interface-endpoint.html
345 https://docs.aws.amazon.com/vpc/latest/privatelink/aws-services-privatelink-support.html
346 https://github.com/kubernetes-sigs/cluster-api-provider-aws

5. (Optional) Configure your cluster to use an existing container registry as a mirror when attempting to pull images. The example below is for AWS ECR:
   ⚠ If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1606) page for more information.

   ⚠ **IMPORTANT:** The AMI must be created by the konvoy-image-builder (see page 1626) project in order to use the registry mirror feature.

   ```
   export REGISTRY_URL=<ecr-registry-URI>
   ```

   - `REGISTRY_URL` : the address of an existing registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
   - NOTE: Other local registries may use the options below:
     - JFrog - `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
     - `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
     - `REGISTRY_PASSWORD` : optional if username is not set.
6. Create a Kubernetes cluster. The following example shows a common configuration. See dkp create cluster aws (see page 1856) reference for the full list of cluster creation options:

⚠ DKP uses local static provisioner as the default storage provider (see page 110). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[347] compatible storage that is suitable for production.

- You can choose from any of the storage options[348] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[349].

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--with-aws-bootstrap-credentials=true \
--vpc-id=${AWS_VPC_ID} \
--ami=${AWS_AMI_ID} \
--subnet-ids=${AWS_SUBNET_IDS} \
```

347 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
348 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
349 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

```
--internal-load-balancer=true \
--additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
--registry-mirror-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD} \
--self-managed
```

Flatcar OS

Flatcar OS (see page 1183) use `--os-hint` to instruct the bootstrap cluster to make some changes related to the installation paths:

```
--os-hint flatcar
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

6. After cluster creation, create the node pool after cluster creation:

```
dkp create nodepool aws -c ${CLUSTER_NAME} \
--instance-type p2.xlarge \
--ami-id=${AMI_ID_FROM_KIB} \
--replicas=1 ${NODEPOOL_NAME} \
--kubeconfig=${CLUSTER_NAME}.conf
```

To understand how this process works step by step, you can find a customizable Create a New AWS Cluster[350] under Custom Installation and Additional Infrastructure Tools (see page 1050) .

4.11.6.5.1 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.11.6.5.2 Next Step:

AWS Air-gapped GPU: Install Kommander (see page 401)

350 https://d2iq.atlassian.net/wiki/pages/createpage.action?fromPageId=161842658&linkCreation=true&spaceKey=DENT&title=AWS+Cluster+Creation+Choices

4.11.6.6 AWS Air-gapped GPU: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

4.11.6.6.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry: Air-gapped Environments (see page 1536).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

4.11.6.6.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init --airgapped > kommander.yaml
   ```

4. *If required*, **customize** your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include:
   - Custom Domains and Certificates
   - HTTP proxy
   - External Load Balancer
   - GPU utilization, etc.
   - Rook Ceph customization for Pre-provisioned environments (see page 1541)
5. *If required*: If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
...
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true
...
```

6. Expand **one** of the following sets of instructions, depending on your license and application environments:

4.11.6.6.1.2 Enable GPU Resources

1. Append the following to the apps section in the `kommander.yaml` file to enable Nvidia platform services.

```
apps:
  nvidia-gpu-operator:
   enabled: true
```

2. Append the correct Nvidia Toolkit version based on your OS:

   The **NVIDIA Container Toolkit** allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

   Centos 7.9/RHEL 7.9:
   If you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.14.6-centos7
```

   RHEL 8.4/8.6 and SLES 15 SP3
   If you're using RHEL 8.4/8.6 or SLES 15 SP3 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.14.6-ubi8
```

Ubuntu 18.04 and 20.04
If you're using Ubuntu 18.04 or 20.04 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.14.6-ubuntu20.04
```

4.11.6.6.1.3 Enable DKP Catalog Applications and Install Kommander in an Air-gapped Environment

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add the following values to enable DKP Catalog Applications:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
```

```
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./dkp-catalog-applications-v2.8.1.tar.gz
```

⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.1.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.1.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.8.1.tar.gz
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.11.6.6.1.4 Next Step:

AWS Air-gapped GPU: Verify Install and Log in the UI (see page 404)

4.11.6.7 AWS Air-gapped GPU: Verify Install and Log in the UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.11.6.7.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

4.11.6.7.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider

- Temporarily Disable an Identity Provider
- Create Groups

4.11.6.7.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

For more GPU information (see page 998), see the section for GPU under Day 2 - Cluster Operations Management.

4.11.6.7.4 Next Step:

AWS Air-gapped GPU: Subsequent New Clusters (see page 407)

4.11.6.8 AWS Air-gapped GPU: Create a Managed Cluster Using the DKP CLI

Enterprise Gov Advanced

In previous steps, you created a new Management cluster, which is self-managed. When you use these steps to create new Managed clusters, they will become Attached clusters under your Management Cluster.

When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

4.11.6.8.1 Create New Managed Air-gapped GPU Cluster with the CLI

The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.

If you do not set a workspace, it will be created in the `default` workspace, and you need to take additional steps to attach to a workspace later. For instructions on how to do this, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster). (see page 784).

Your new, managed cluster needs to be part of either a new Workspace or an existing Workspace under a management cluster. Create or locate a workspace name to get started with your managed cluster.

4.11.6.8.2 Make New Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.11.6.8.3 Name Your Cluster

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

   ```
   export MANAGED_CLUSTER_NAME=<aws-additional>
   ```

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[351] for more naming information.

3. Execute this command to create your Kubernetes cluster using any relevant flags but note that the `--self-managed` is absent so that this cluster can be attached to your previously created Management Cluster.

   - Execute this command to create a cluster with a GPU AMI:

351 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
dkp create cluster aws --cluster-name=${MANAGED_CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--namespace ${WORKSPACE_NAMESPACE}
--with-aws-bootstrap-credentials=true \
--vpc-id=${AWS_VPC_ID} \
--ami=${AWS_AMI_ID} \
--subnet-ids=${AWS_SUBNET_IDS} \
--internal-load-balancer=true \
--additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
--registry-mirror-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD} \
--kubeconfig=<management-cluster-kubeconfig-path> \
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

- Create the node pool after cluster creation:

```
dkp create nodepool aws -c ${CLUSTER_NAME} \
--instance-type p2.xlarge \
--ami-id=${AMI_ID_FROM_KIB} \
--replicas=1 ${NODEPOOL_NAME} \
--kubeconfig=${CLUSTER_NAME}.conf
```

4.11.6.8.4 Retrieve the kubeconfig and Explore New AWS Cluster

Follow these steps:

1. Fetch the kubeconfig file with the command:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} --kubeconfig
<management-cluster-kubeconfig-path> -n ${WORKSPACE_NAMESPACE} > $
{MANAGED_CLUSTER_NAME}.conf
```

2. List the nodes with the following command:

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get nodes
```

3. List the pods with the following command:

> **NOTE:** Wait for the Status to move to Ready while `calico-node` pods are being deployed.

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get pods -A
```

4.11.6.8.4.1 Manually Attach a DKP CLI Cluster to the Management Cluster

> These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

   ```
   echo ${MANAGED_CLUSTER_NAME}
   ```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
   NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

   ```
   kubectl get workspaces -A
   ```

5. Set the WORKSPACE_NAMESPACE environment variable:

```
export WORKSPACE_NAMESPACE=<workspace-namespace>
```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

```
kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
template='{{.data.value}}{{ "\n"}}'
```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

4.11.6.8.4.2 Next Step:

Day 2 - Cluster Operations Management (see page 590)

4.12 EKS Install Options

Enterprise Gov Advanced

For an environment that is EKS on the AWS Infrastructure, install options are provided for you in this one location. Remember, there are always more options in the EKS Infrastructure (see page 1259) section under Custom Installation and Additional Infrastructure Tools (see page 1050), but this will get you operative in the most common scenario.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

An EKS cluster cannot be a Management (see page 80) or Essential (see page 81) cluster. To install DKP on your EKS cluster, first ensure you have a Management cluster with DKP and the Kommander component installed, that handles the lifecycle of your EKS cluster.

Refer to this page to check if your OS is a Supported Operating System (see page 67). Then proceed to EKS installation:

- EKS Install (see page 413)

In order to install Kommander, you need to have CAPI components, cert-manager, etc on a self-managed cluster. The CAPI components mean you can control the lifecycle of the cluster, and other clusters. However, because EKS is semi-managed by AWS, the EKS clusters are under AWS control and don't have those components. Therefore, Kommander will not be installed.

4.12.1 EKS Install

Enterprise Gov Advanced

This section provides instructions to install DKP with EKS in an AWS non-air-gapped environment.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

> Ensure that the `KUBECONFIG` environment variable is set to the Management cluster by running `export KUBECONFIG=<Management_cluster_kubeconfig>.conf`.

4.12.1.1 AWS prerequisites

Before you begin using DKP with AWS, you must have:

- A Management cluster with the Kommander component installed.

> An EKS cluster cannot be a Management (see page 80) or Essential (see page 81) cluster. To install DKP on your EKS cluster, first ensure you have a Management cluster with DKP and the Kommander component installed, that handles the lifecycle of your EKS cluster.

- You have a valid AWS account with credentials configured[352] that can manage CloudFormation Stacks, IAM Policies, and IAM Roles.
- You will need to have the AWS CLI utility installed[353].
- Install aws-iam-authenticator[354]. This binary is used to access your cluster using kubectl.

352 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html
353 https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html
354 https://docs.aws.amazon.com/eks/latest/userguide/install-aws-iam-authenticator.html

> In order to install Kommander, you need to have CAPI components, cert-manager, etc on a self-managed cluster. The CAPI components mean you can control the lifecycle of the cluster, and other clusters. However, because EKS is semi-managed by AWS, the EKS clusters are under AWS control and don't have those components. Therefore, Kommander will not be installed and these clusters will be attached to the management cluster.

4.12.1.2 Next Step:

EKS: Minimal User Permission for Cluster Creation (see page 414)

4.12.1.3 EKS: Minimal User Permission for Cluster Creation

Enterprise Gov Advanced

The following is a CloudFormation stack which adds a policy named `eks-bootstrapper` to manage EKS cluster to the `dkp-bootstrapper-role` created by the CloudFormation stack for AWS in the Minimal Permissions and Role to Create Cluster (see page 1156) section.

Consult the Leveraging the Role (see page 1160)section for an example of how to use this role and how a system administrator wants to expose using the permissions.

4.12.1.3.1 EKS CloudFormation stack:

```
AWSTemplateFormatVersion: 2010-09-09
Parameters:
  existingBootstrapperRole:
    Type: CommaDelimitedList
    Description: 'Name of existing minimal role you want to add to add EKS cluster
management permissions to'
    Default: dkp-bootstrapper-role
Resources:
  EKSMinimumPermissions:
    Properties:
      Description: Minimal user policy to manage eks clusters
      ManagedPolicyName: eks-bootstrapper
      PolicyDocument:
        Statement:
          - Action:
              - 'ssm:GetParameter'
            Effect: Allow
            Resource:
              - 'arn:*:ssm:*:*:parameter/aws/service/eks/optimized-ami/*'
          - Action:
```

```
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:*:iam::*:role/aws-service-role/eks.amazonaws.com/
AWSServiceRoleForAmazonEKS
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks-nodegroup.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:*:iam::*:role/aws-service-role/eks-nodegroup.amazonaws.com/
AWSServiceRoleForAmazonEKSNodegroup
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks-fargate.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:aws:iam::*:role/aws-service-role/eks-fargate-pods.amazonaws.com/
AWSServiceRoleForAmazonEKSForFargate
          - Action:
              - 'iam:GetRole'
              - 'iam:ListAttachedRolePolicies'
            Effect: Allow
            Resource:
              - 'arn:*:iam::*:role/*'
          - Action:
              - 'iam:GetPolicy'
            Effect: Allow
            Resource:
              - 'arn:aws:iam::aws:policy/AmazonEKSClusterPolicy'
          - Action:
              - 'eks:DescribeCluster'
              - 'eks:ListClusters'
              - 'eks:CreateCluster'
              - 'eks:TagResource'
              - 'eks:UpdateClusterVersion'
              - 'eks:DeleteCluster'
              - 'eks:UpdateClusterConfig'
              - 'eks:UntagResource'
              - 'eks:UpdateNodegroupVersion'
              - 'eks:DescribeNodegroup'
              - 'eks:DeleteNodegroup'
```

```
            - 'eks:UpdateNodegroupConfig'
            - 'eks:CreateNodegroup'
            - 'eks:AssociateEncryptionConfig'
            - 'eks:ListIdentityProviderConfigs'
            - 'eks:AssociateIdentityProviderConfig'
            - 'eks:DescribeIdentityProviderConfig'
            - 'eks:DisassociateIdentityProviderConfig'
          Effect: Allow
          Resource:
            - 'arn:*:eks:*:*:cluster/*'
            - 'arn:*:eks:*:*:nodegroup/*/*/*'
        - Action:
            - 'ec2:AssociateVpcCidrBlock'
            - 'ec2:DisassociateVpcCidrBlock'
            - 'eks:ListAddons'
            - 'eks:CreateAddon'
            - 'eks:DescribeAddonVersions'
            - 'eks:DescribeAddon'
            - 'eks:DeleteAddon'
            - 'eks:UpdateAddon'
            - 'eks:TagResource'
            - 'eks:DescribeFargateProfile'
            - 'eks:CreateFargateProfile'
            - 'eks:DeleteFargateProfile'
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - 'iam:PassRole'
          Condition:
            StringEquals:
              'iam:PassedToService': eks.amazonaws.com
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - 'kms:CreateGrant'
            - 'kms:DescribeKey'
          Condition:
            'ForAnyValue:StringLike':
              'kms:ResourceAliases': alias/cluster-api-provider-aws-*
          Effect: Allow
          Resource:
            - '*'
      Version: 2012-10-17
    Roles: !Ref existingBootstrapperRole
  Type: 'AWS::IAM::ManagedPolicy'
```

If your role is not named `dkp-bootstrapper-role` change the parameter on line 6 of the file.

To create the resources in the cloudformation stack, copy the contents above into a file. Before executing the following command, replace `MYFILENAME.yaml` and `MYSTACKNAME` with the intended values for your system:

```
aws cloudformation create-stack --template-body=file://MYFILENAME.yaml --stack-
name=MYSTACKNAME --capabilities CAPABILITY_NAMED_IAM
```

4.12.1.3.2 Next Step:

EKS: Cluster IAM Policies and Roles (see page 417)

4.12.1.4 EKS: Cluster IAM Policies and Roles

Enterprise Gov Advanced

This section guides a DKP user in creating IAM Policies and Instance Profiles that governs who has access to the cluster. The IAM Role is used by the cluster's control plane and worker nodes using the provided AWS CloudFormation Stack specific to EKS. This CloudFormation Stack has additional permissions that are used to delegate access roles for other users.

4.12.1.4.1 Prerequisites from AWS:

- The user you delegate from your role must have a minimum set of permissions, see User Roles and Instance Profiles (see page 1156) page for AWS.
- Create the Cluster IAM Policies (see page 1162) in your AWS account.

4.12.1.4.1.1 EKS IAM Artifacts

Policies

- `controllers-eks.cluster-api-provider-aws.sigs.k8s.io` - enumerates the Actions required by the workload cluster to create and modify EKS clusters in the user's AWS Account. It is attached to the existing `control-plane.cluster-api-provider-aws.sigs.k8s.io` role

- `eks-nodes.cluster-api-provider-aws.sigs.k8s.io` - enumerates the Actions required by the EKS workload cluster's worker machines. It is attached to the existing `nodes.cluster-api-provider-aws.sigs.k8s.io`

Roles

- `eks-controlplane.cluster-api-provider-aws.sigs.k8s.io` - is the Role associated with EKS cluster control planes

NOTE: `control-plane.cluster-api-provider-aws.sigs.k8s.io` and `nodes.cluster-api-provider-aws.sigs.k8s.io` roles were created by Cluster IAM Policies and Roles (see page 1162) in AWS.

Below is a CloudFormation stack[355] that includes IAM policies and roles required to setup EKS Clusters:

```
AWSTemplateFormatVersion: 2010-09-09
Parameters:
  existingControlPlaneRole:
    Type: CommaDelimitedList
    Description: 'Names of existing Control Plane Role you want to add to the newly
created EKS Managed Policy for AWS cluster API controllers'
    Default: control-plane.cluster-api-provider-aws.sigs.k8s.io
  existingNodeRole:
    Type: CommaDelimitedList
    Description: 'ARN of the Nodes Managed Policy to add to the role for nodes'
    Default: nodes.cluster-api-provider-aws.sigs.k8s.io
Resources:
  AWSIAMManagedPolicyControllersEKS:
    Properties:
      Description: For the Kubernetes Cluster API Provider AWS Controllers
      ManagedPolicyName: controllers-eks.cluster-api-provider-aws.sigs.k8s.io
      PolicyDocument:
        Statement:
          - Action:
              - 'ssm:GetParameter'
            Effect: Allow
            Resource:
              - 'arn:*:ssm:*:*:parameter/aws/service/eks/optimized-ami/*'
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
```

355 https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html

```
                'iam:AWSServiceName': eks.amazonaws.com
              Effect: Allow
              Resource:
                - >-
                  arn:*:iam::*:role/aws-service-role/eks.amazonaws.com/
AWSServiceRoleForAmazonEKS
            - Action:
                - 'iam:CreateServiceLinkedRole'
              Condition:
                StringLike:
                  'iam:AWSServiceName': eks-nodegroup.amazonaws.com
              Effect: Allow
              Resource:
                - >-
                  arn:*:iam::*:role/aws-service-role/eks-nodegroup.amazonaws.com/
AWSServiceRoleForAmazonEKSNodegroup
            - Action:
                - 'iam:CreateServiceLinkedRole'
              Condition:
                StringLike:
                  'iam:AWSServiceName': eks-fargate.amazonaws.com
              Effect: Allow
              Resource:
                - >-
                  arn:aws:iam::*:role/aws-service-role/eks-fargate-pods.amazonaws.com/
AWSServiceRoleForAmazonEKSForFargate
            - Action:
                - 'iam:GetRole'
                - 'iam:ListAttachedRolePolicies'
              Effect: Allow
              Resource:
                - 'arn:*:iam::*:role/*'
            - Action:
                - 'iam:GetPolicy'
              Effect: Allow
              Resource:
                - 'arn:aws:iam::aws:policy/AmazonEKSClusterPolicy'
            - Action:
                - 'eks:DescribeCluster'
                - 'eks:ListClusters'
                - 'eks:CreateCluster'
                - 'eks:TagResource'
                - 'eks:UpdateClusterVersion'
                - 'eks:DeleteCluster'
                - 'eks:UpdateClusterConfig'
                - 'eks:UntagResource'
                - 'eks:UpdateNodegroupVersion'
                - 'eks:DescribeNodegroup'
                - 'eks:DeleteNodegroup'
                - 'eks:UpdateNodegroupConfig'
                - 'eks:CreateNodegroup'
                - 'eks:AssociateEncryptionConfig'
```

```
            - 'eks:ListIdentityProviderConfigs'
            - 'eks:AssociateIdentityProviderConfig'
            - 'eks:DescribeIdentityProviderConfig'
            - 'eks:DisassociateIdentityProviderConfig'
          Effect: Allow
          Resource:
            - 'arn:*:eks:*:*:cluster/*'
            - 'arn:*:eks:*:*:nodegroup/*/*/*'
        - Action:
            - 'ec2:AssociateVpcCidrBlock'
            - 'ec2:DisassociateVpcCidrBlock'
            - 'eks:ListAddons'
            - 'eks:CreateAddon'
            - 'eks:DescribeAddonVersions'
            - 'eks:DescribeAddon'
            - 'eks:DeleteAddon'
            - 'eks:UpdateAddon'
            - 'eks:TagResource'
            - 'eks:DescribeFargateProfile'
            - 'eks:CreateFargateProfile'
            - 'eks:DeleteFargateProfile'
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - 'iam:PassRole'
          Condition:
            StringEquals:
              'iam:PassedToService': eks.amazonaws.com
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - 'kms:CreateGrant'
            - 'kms:DescribeKey'
          Condition:
            'ForAnyValue:StringLike':
              'kms:ResourceAliases': alias/cluster-api-provider-aws-*
          Effect: Allow
          Resource:
            - '*'
      Version: 2012-10-17
    Roles: !Ref existingControlPlaneRole
  Type: 'AWS::IAM::ManagedPolicy'
AWSIAMManagedEKSNodesPolicy:
  Properties:
    Description: Additional Policies to nodes role to work for EKS
    ManagedPolicyName: eks-nodes.cluster-api-provider-aws.sigs.k8s.io
    PolicyDocument:
      Statement:
        - Action:
            - "ec2:AssignPrivateIpAddresses"
```

```
            - "ec2:AttachNetworkInterface"
            - "ec2:CreateNetworkInterface"
            - "ec2:DeleteNetworkInterface"
            - "ec2:DescribeInstances"
            - "ec2:DescribeTags"
            - "ec2:DescribeNetworkInterfaces"
            - "ec2:DescribeInstanceTypes"
            - "ec2:DetachNetworkInterface"
            - "ec2:ModifyNetworkInterfaceAttribute"
            - "ec2:UnassignPrivateIpAddresses"
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - ec2:CreateTags
          Effect: Allow
          Resource:
            - arn:aws:ec2:*:*:network-interface/*
        - Action:
            - "ec2:DescribeInstances"
            - "ec2:DescribeInstanceTypes"
            - "ec2:DescribeRouteTables"
            - "ec2:DescribeSecurityGroups"
            - "ec2:DescribeSubnets"
            - "ec2:DescribeVolumes"
            - "ec2:DescribeVolumesModifications"
            - "ec2:DescribeVpcs"
            - "eks:DescribeCluster"
          Effect: Allow
          Resource:
            - '*'
      Version: 2012-10-17
    Roles: !Ref existingNodeRole
  Type: 'AWS::IAM::ManagedPolicy'
AWSIAMRoleEKSControlPlane:
  Properties:
    AssumeRolePolicyDocument:
      Statement:
        - Action:
            - 'sts:AssumeRole'
          Effect: Allow
          Principal:
            Service:
              - eks.amazonaws.com
      Version: 2012-10-17
    ManagedPolicyArns:
      - 'arn:aws:iam::aws:policy/AmazonEKSClusterPolicy'
    RoleName: eks-controlplane.cluster-api-provider-aws.sigs.k8s.io
  Type: 'AWS::IAM::Role'
```

To create the resources in the CloudFormation stack, copy the contents above into a file and execute the following command after replacing `MYFILENAME.yaml` and `MYSTACKNAME` with the intended values:

```
 aws cloudformation create-stack --template-body=file://MYFILENAME.yaml --stack-
name=MYSTACKNAME --capabilities  CAPABILITY_NAMED_IAM
```

Add EKS CSI Policy

AWS CloudFormation does not support attaching an existing IAM Policy to an existing IAM Role. Add the necessary IAM policy to your worker instance profile using the `aws` CLI:

```
aws iam attach-role-policy --role-name nodes.cluster-api-provider-aws.sigs.k8s.io --
policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy
```

4.12.1.4.1.2 Next Step:

EKS: Create an Image (see page 422)

4.12.1.5 EKS: Create an Image

Enterprise Gov Advanced

AWS EKS best practices discourage building custom images. The Amazon EKS Optimized AMI[356] is the preferred way to deploy containers for EKS. If the image is customized, it breaks some of the autoscaling and security capabilities of EKS.

4.12.1.5.1 Next Step:

EKS: Cluster Creation (see page 422)

4.12.1.6 EKS: Create an EKS Cluster

Enterprise Gov Advanced

When creating a Managed (see page 80) cluster on your EKS infrastructure, you can choose from multiple configuration types. The steps for creating and accessing your cluster are listed below after setting minimal permissions (see page 414) and creating IAM Policies and Roles (see page 417). For more information about custom EKS configurations, refer to the EKS Infrastructure (see page 1259) under Custom Installation and Additional Infrastructure Tools (see page 1050).

356 https://docs.aws.amazon.com/eks/latest/userguide/eks-optimized-amis.html

4.12.1.6.1 Access Cluster

- Use previously installed aws-iam-authenticator[357] to access your cluster using kubectl. Amazon EKS uses IAM to provide authentication to your Kubernetes cluster.
- Export the AWS region where you want to deploy the cluster:

```
export AWS_REGION=us-west-2
```

- Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

```
export AWS_PROFILE=<profile>
```

See the AWS site for more information about AWS credentials.[358]

Ensure that the `KUBECONFIG` environment variable is set to the Management cluster by running `export KUBECONFIG=<Management_cluster_kubeconfig>.conf`.

4.12.1.6.2 Name Your Cluster

Follow these steps:

1. Give your cluster a unique name suitable for your environment.

   In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

2. Set the environment variable:

```
export CLUSTER_NAME=<aws-example>
```

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[359] for more naming information.

357 https://docs.aws.amazon.com/eks/latest/userguide/install-aws-iam-authenticator.html
358 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html
359 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

4.12.1.6.3 Create a New EKS Kubernetes Cluster from the CLI

If you prefer to work in the shell, you can continue by creating a new cluster following these steps. If you prefer to log in to the DKP UI, you can create a new cluster from there using the steps on this page: Create an EKS Cluster from the DKP UI (see page 1278)

> By default, the control-plane Nodes will be created in 3 different Availability Zones. However, the default worker Nodes will reside in a single zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Set the environment variable to the name you assigned this cluster.

   ```
   export CLUSTER_NAME=eks-example
   ```

2. Make sure your AWS credentials are up-to-date. Refresh the credentials command is only necessary if you are using Static Credentials (see page 0) (Access Keys) otherwise, if you are using role-based authentication on a bastion host, proceed to step 3:

   ```
   dkp update bootstrap credentials aws
   ```

3. Create the cluster:

   ```
   dkp create cluster eks \
     --cluster-name=${CLUSTER_NAME} \
     --additional-tags=owner=$(whoami)
   ```

   ⚠️ **NOTE**: If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

   ⚠️ **Optional flag for ECR**: Configure your cluster to use an existing local registry as a mirror when attempting to pull images. Below is an AWS ECR example:

   ```
   export --registry-mirror-url-string=<YOUR_ECR_URL>
   ```

   - `REGISTRY-MIRROR-URL-STRING`: the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling

images. Users can still pull their own images from ECR directly or use ECR as a mirror. See CLI commands for `--registry-mirror` flag here: dkp create cluster eks (see page 1854)

```
Generating cluster resources
cluster.cluster.x-k8s.io/eks-example created
awsmanagedcontrolplane.controlplane.cluster.x-k8s.io/eks-example-control-plane
 created
machinedeployment.cluster.x-k8s.io/eks-example-md-0 created
awsmachinetemplate.infrastructure.cluster.x-k8s.io/eks-example-md-0 created
eksconfigtemplate.bootstrap.cluster.x-k8s.io/eks-example-md-0 created
clusterresourceset.addons.cluster.x-k8s.io/calico-cni-installation-eks-example
 created
configmap/calico-cni-installation-eks-example created
configmap/tigera-operator-eks-example created
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-eks-example
 created
configmap/cluster-autoscaler-eks-example created
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-eks-example
 created
configmap/node-feature-discovery-eks-example created
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-eks-example
 created
configmap/nvidia-feature-discovery-eks-example created
```

4. Inspect or edit the cluster objects:

   Use your favorite editor.

   NOTE: Editing the cluster objects requires some understanding of Cluster API. Edits can prevent the cluster from deploying successfully.

   The objects are Custom Resources[360] defined by Cluster API components, and they belong in three different categories:

   a. **Cluster**
      A *Cluster* object has references to the infrastructure-specific and control plane objects. Because this is an AWS cluster, there is an *AWSCluster* object that describes the infrastructure-specific cluster properties. Here, this means the AWS region, the VPC ID, subnet IDs, and security group rules required by the Pod network implementation.

   b. **Control Plane**
      A *AWSManagedControlPlane* object describes the control plane, which is the group of machines that run the Kubernetes control plane components, which include the etcd distributed database, the API server, the core controllers, and the scheduler. The object describes the configuration for these components. The object also has a reference to an infrastructure-specific object that describes the properties of all control plane machines.

   c. **Node Pool**
      A Node Pool is a collection of machines with identical properties. For example, a cluster might have one Node Pool with large memory capacity, another Node Pool with GPU support. Each Node Pool is described by three objects: The MachinePool references an object that

360 https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/

describes the configuration of Kubernetes components (for example, kubelet) deployed on each node pool machine, and an infrastructure-specific object that describes the properties of all node pool machines. Here, it references a *KubeadmConfigTemplate*, and an *AWSMachineTemplate* object, which describes the instance type, the type of disk used, the size of the disk, among other properties.

For in-depth documentation about the objects, read Concepts[361] in the Cluster API Book.

5. Wait for the cluster control-plane to be ready:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=20m
```

```
cluster.cluster.x-k8s.io/eks-example condition met
```

The `READY` status will become `True` after the cluster control-plane becomes ready in one of the following steps.

6. After the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. Konvoy provides a command to describe the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/eks-example                                                 True
10m
├─ControlPlane - AWSManagedControlPlane/eks-example-control-plane  True
10m
└─Workers
  └─MachineDeployment/eks-example-md-0                              True
26s
    ├─Machine/eks-example-md-0-78fcd7c7b7-66ntt                     True
84s
    ├─Machine/eks-example-md-0-78fcd7c7b7-b9qmc                     True
84s
    ├─Machine/eks-example-md-0-78fcd7c7b7-v5vfq                     True
84s
    └─Machine/eks-example-md-0-78fcd7c7b7-zl6m2                     True
84s
```

361 https://cluster-api.sigs.k8s.io/user/concepts.html

7. As they progress, the controllers also create Events. List the Events using this command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, the example uses `grep`. It is also possible to use separate commands to get Events for specific objects. For example, `kubectl get events --field-selector involvedObject.kind="AWSCluster"` and `kubectl get events --field-selector involvedObject.kind="AWSMachine"`.

```
46m         Normal   SuccessfulCreateVPC
awsmanagedcontrolplane/eks-example-control-plane   Created new managed VPC
"vpc-05e775702092abf09"
46m         Normal   SuccessfulSetVPCAttributes
awsmanagedcontrolplane/eks-example-control-plane   Set managed VPC attributes
for "vpc-05e775702092abf09"
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0419dd3f2dfd95ff8"
46m         Normal   SuccessfulModifySubnetAttributes
awsmanagedcontrolplane/eks-example-control-plane   Modified managed Subnet
"subnet-0419dd3f2dfd95ff8" attributes
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0e724b128e3113e47"
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-06b2b31ea6a8d3962"
46m         Normal   SuccessfulModifySubnetAttributes
awsmanagedcontrolplane/eks-example-control-plane   Modified managed Subnet
"subnet-06b2b31ea6a8d3962" attributes
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0626ce238be32bf98"
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0f53cf59f83177800"
46m         Normal   SuccessfulModifySubnetAttributes
awsmanagedcontrolplane/eks-example-control-plane   Modified managed Subnet
"subnet-0f53cf59f83177800" attributes
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0878478f6bbf153b2"
46m         Normal   SuccessfulCreateInternetGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Internet
Gateway "igw-09fb52653949d4579"
46m         Normal   SuccessfulAttachInternetGateway
awsmanagedcontrolplane/eks-example-control-plane   Internet Gateway
"igw-09fb52653949d4579" attached to VPC "vpc-05e775702092abf09"
```

```
46m         Normal   SuccessfulCreateNATGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new NAT Gateway
"nat-06356aac28079952d"
46m         Normal   SuccessfulCreateNATGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new NAT Gateway
"nat-0429d1cd9d956bf35"
46m         Normal   SuccessfulCreateNATGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new NAT Gateway
"nat-059246bcc9d4e88e7"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-01689c719c484fd3c"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-01689c719c484fd3c" with subnet "subnet-0419dd3f2dfd95ff8"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-065af81b9752eeb69"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-065af81b9752eeb69" with subnet "subnet-0e724b128e3113e47"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-03eeff810a89afc98"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-03eeff810a89afc98" with subnet "subnet-06b2b31ea6a8d3962"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-0fab36f8751fdee73"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-0fab36f8751fdee73" with subnet "subnet-0626ce238be32bf98"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-0e5c9c7bbc3740a0f"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-0e5c9c7bbc3740a0f" with subnet "subnet-0f53cf59f83177800"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-0bf58eb5f73c387af"
```

```
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-0bf58eb5f73c387af" with subnet "subnet-0878478f6bbf153b2"
46m         Normal   SuccessfulCreateSecurityGroup
awsmanagedcontrolplane/eks-example-control-plane   Created managed
SecurityGroup "sg-0b045c998a120a1b2" for Role "node-eks-additional"
46m         Normal   InitiatedCreateEKSControlPlane
awsmanagedcontrolplane/eks-example-control-plane   Initiated creation of a new
 EKS control plane default_eks-example-control-plane
37m         Normal   SuccessfulCreateEKSControlPlane
awsmanagedcontrolplane/eks-example-control-plane   Created new EKS control
plane default_eks-example-control-plane
37m         Normal   SucessfulCreateKubeconfig
awsmanagedcontrolplane/eks-example-control-plane   Created kubeconfig for
 cluster "eks-example"
37m         Normal   SucessfulCreateUserKubeconfig
awsmanagedcontrolplane/eks-example-control-plane   Created user kubeconfig for
 cluster "eks-example"
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-4t9nc                  Created new node instance
with id "i-0aecc1897c93df740"
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-4t9nc                  AWS Secret entries
containing userdata deleted
26m         Normal   SuccessfulSetNodeRef                              machine/
eks-example-md-0-78fcd7c7b7-fn7x9          ip-10-0-88-24.us-west-2.compute.inte
rnal
26m         Normal   SuccessfulSetNodeRef                              machine/
eks-example-md-0-78fcd7c7b7-g64nv          ip-10-0-110-219.us-west-2.compute.in
ternal
26m         Normal   SuccessfulSetNodeRef                              machine/
eks-example-md-0-78fcd7c7b7-gwc5j          ip-10-0-101-161.us-west-2.compute.in
ternal
26m         Normal   SuccessfulSetNodeRef                              machine/
eks-example-md-0-78fcd7c7b7-j58s4          ip-10-0-127-49.us-west-2.compute.int
ernal
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-fn7x9"
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-g64nv"
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-j58s4"
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-gwc5j"
```

```
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-7whkv                  Created new node instance
with id "i-06dfc0466b8f26695"
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-7whkv                  AWS Secret entries
containing userdata deleted
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-ttgzv                  Created new node instance
with id "i-0544fce0350fd41fb"
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-ttgzv                  AWS Secret entries
containing userdata deleted
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-v2hrf                  Created new node instance
with id "i-0498906edde162e59"
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-v2hrf                  AWS Secret entries
containing userdata deleted
46m         Normal   SuccessfulCreate
machinedeployment/eks-example-md-0                 Created MachineSet "eks-
example-md-0-78fcd7c7b7"
```

4.12.1.6.4 Known Limitations

Be aware of these limitations in the current release of Konvoy.

- The Konvoy version used to create a workload cluster must match the Konvoy version used to delete a workload cluster.
- EKS clusters cannot be Self-managed.
- Konvoy supports deploying one workload cluster.
- Konvoy generates a set of objects for one Node Pool.
- Konvoy does not validate edits to cluster objects.

4.12.1.6.4.1 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.12.1.6.4.2 Next Step:

EKS: Grant Cluster Access (see page 431)

4.12.1.7 EKS: Grant Cluster Access

Enterprise Gov Advanced

4.12.1.7.1 How to Grant EKS Cluster Access

You can access your cluster using AWS IAM roles in the dashboard. When you create an EKS cluster, the IAM entity is granted `system:masters` permissions in Kubernetes Role Based Access Control (RBAC) configuration.[362]

> More information about the configuration of the EKS control plane can be found on the EKS Cluster IAM Policies and Roles (see page 1266) page.

If the EKS cluster was created as a cluster using a self-managed AWS cluster that uses IAM Instance Profiles, you will need to modify the `IAMAuthenticatorConfig` field in the `AWSManagedControlPlane` API object to allow other IAM entities to access the EKS workload cluster. Follow the steps below:

1. Run the following command with your `KUBECONFIG` configured to select the self-managed cluster previously used to create the workload EKS cluster. Ensure you substitute `${CLUSTER_NAME}` and `${CLUSTER_NAMESPACE}` with their corresponding values for your cluster.

   ```
   kubectl edit awsmanagedcontrolplane ${CLUSTER_NAME}-control-plane -n $
   {CLUSTER_NAMESPACE}
   ```

2. Edit the `IamAuthenticatorConfig` field with the IAM Role to the corresponding Kubernetes Role. In this example, the IAM role `arn:aws:iam::111122223333:role/PowerUser` is granted the cluster role `system:masters`. Note that this example uses example AWS resource ARNs[363], so these values should be substituted for real values in the corresponding AWS account.

   ```
   iamAuthenticatorConfig:
       mapRoles:
         - groups:
           - system:bootstrappers
           - system:nodes
           rolearn: arn:aws:iam::111122223333:role/my-node-role
   ```

362 https://kubernetes.io/docs/reference/access-authn-authz/rbac/
363 https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html

```
    username: system:node:{{EC2PrivateDNSName}}
  - groups:
    - system:masters
    rolearn: arn:aws:iam::111122223333:role/PowerUser
    username: admin
```

For further instructions on changing or assigning `roles` or `clusterroles` to which you can map IAM users or roles, see Amazon Enabling IAM access to your cluster[364].

4.12.1.7.2 Next Step:

EKS: Retrieve kubeconfig for EKS Cluster (see page 432)

4.12.1.8 EKS: Retrieve kubeconfig for EKS Cluster

Enterprise Gov Advanced

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster.

Before you start, make sure you have created a workload cluster, as described in EKS: Create an EKS Cluster (see page 422) .

4.12.1.8.1 Explore the New Kubernetes Cluster

Follow these steps:

1. Get a kubeconfig file for the workload cluster:

   When the workload cluster is created, the cluster lifecycle services generate a kubeconfig file for the workload cluster, and write it to a *Secret*. The kubeconfig file is scoped to the cluster administrator.

   Get the kubeconfig from the *Secret*, and write it to a file, using this command:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

2. List the Nodes using this command:

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
   ```

   Output will be similar to:

   ```
   NAME                                          STATUS   ROLES    AGE   VERSION
   ip-10-0-122-211.us-west-2.compute.internal   Ready    <none>   35m   v1.27.12-
   eks-ae9a62a
   ```

364 https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html

```
ip-10-0-127-74.us-west-2.compute.internal    Ready    <none>   35m   v1.27.12-
eks-ae9a62a
ip-10-0-71-155.us-west-2.compute.internal    Ready    <none>   35m   v1.27.12-
eks-ae9a62a
ip-10-0-93-47.us-west-2.compute.internal     Ready    <none>   35m   v1.27.12-
eks-ae9a62a
```

It may take a few minutes for the Status to move to `Ready` while the Pod network is deployed. The Nodes' Status should change to Ready soon after the `calico-node` DaemonSet Pods are Ready.

3. List the Pods using this command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get --all-namespaces pods
```

Output will be similar to:

```
NAMESPACE                 NAME                                                READY
STATUS     RESTARTS       AGE
calico-system             calico-kube-controllers-7d6749878f-ccsx9            1/1
Running    0              34m
calico-system             calico-node-2r6l8                                   1/1
Running    0              34m
calico-system             calico-node-5pdlb                                   1/1
Running    0              34m
calico-system             calico-node-n24hh                                   1/1
Running    0              34m
calico-system             calico-node-qrh7p                                   1/1
Running    0              34m
calico-system             calico-typha-7bbcb87696-7pk45                       1/1
Running    0              34m
calico-system             calico-typha-7bbcb87696-t4c8r                       1/1
Running    0              34m
calico-system             csi-node-driver-bz48k                               2/2
Running    0              34m
calico-system             csi-node-driver-k5mmk                               2/2
Running    0              34m
calico-system             csi-node-driver-nvcck                               2/2
Running    0              34m
calico-system             csi-node-driver-x4xnh                               2/2
Running    0              34m
kube-system               aws-node-2xp86                                      1/1
Running    0              35m
```

```
kube-system                aws-node-5f2kx                                    1/1
Running    0               35m
kube-system                aws-node-6lzm7                                    1/1
Running    0               35m
kube-system                aws-node-pz8c6                                    1/1
Running    0               35m
kube-system                cluster-autoscaler-789d86b489-sz9x2               0/1
Init:0/1   0               36m
kube-system                coredns-57ff979f67-pk5cg                          1/1
Running    0               75m
kube-system                coredns-57ff979f67-sf2j9                          1/1
Running    0               75m
kube-system                ebs-csi-controller-5f6bd5d6dc-bplwm               6/6
Running    0               36m
kube-system                ebs-csi-controller-5f6bd5d6dc-dpjt7               6/6
Running    0               36m
kube-system                ebs-csi-node-7hmm5                                3/3
Running    0               35m
kube-system                ebs-csi-node-l4vfh                                3/3
Running    0               35m
kube-system                ebs-csi-node-mfr7c                                3/3
Running    0               35m
kube-system                ebs-csi-node-v8krq                                3/3
Running    0               35m
kube-system                kube-proxy-7fc5x                                  1/1
Running    0               35m
kube-system                kube-proxy-vvkmk                                  1/1
Running    0               35m
kube-system                kube-proxy-x6hcc                                  1/1
Running    0               35m
kube-system                kube-proxy-x8frb                                  1/1
Running    0               35m
kube-system                snapshot-controller-8ff89f489-4cfxv               1/1
Running    0               36m
kube-system                snapshot-controller-8ff89f489-78gg8               1/1
Running    0               36m
node-feature-discovery     node-feature-discovery-master-7d5985467-52fcn     1/1
Running    0               36m
node-feature-discovery     node-feature-discovery-worker-88hr7               1/1
Running    0               34m
node-feature-discovery     node-feature-discovery-worker-h95nq               1/1
Running    0               35m
node-feature-discovery     node-feature-discovery-worker-lfghg               1/1
Running    0               34m
node-feature-discovery     node-feature-discovery-worker-prc8p               1/1
Running    0               35m
tigera-operator            tigera-operator-6dcd98c8ff-k97hq                  1/1
Running    0               36m
```

4.12.1.8.1.1 Next Step:

EKS: Attach a Cluster (see page 435)

4.12.1.9 EKS: Attach a Cluster

Enterprise Gov Advanced

You can attach existing Kubernetes clusters to the Management Cluster. After attaching the cluster, you can use the UI to examine and manage (see page 774) this cluster. The following procedure shows how to attach an existing Amazon Elastic Kubernetes Service (EKS) cluster.

> This procedure assumes you have an existing and spun up Amazon EKS cluster(s) with administrative privileges. Refer to the Amazon EKS[365] for setup and configuration information.

- Install aws-iam-authenticator[366]. This binary is used to access your cluster using kubectl.

4.12.1.9.1 Attach a Pre-existing EKS Cluster

Ensure that the `KUBECONFIG` environment variable is set to the Management cluster before attaching by running:

```
export KUBECONFIG=<Management_cluster_kubeconfig>.conf
```

4.12.1.9.1.1 Access your EKS clusters

1. Ensure you are connected to your EKS clusters. Enter the following commands for each of your clusters:

   ```
   kubectl config get-contexts
   kubectl config use-context <context for first eks cluster>
   ```

2. Confirm `kubectl` can access the EKS cluster:

   ```
   kubectl get nodes
   ```

365 https://aws.amazon.com/eks/
366 https://docs.aws.amazon.com/eks/latest/userguide/install-aws-iam-authenticator.html

4.12.1.9.1.2 Create a `kubeconfig` File

To get started, ensure you have kubectl[367] set up and configured with ClusterAdmin[368] for the cluster you want to connect to Kommander.

1. Create the necessary service account:

   ```
   kubectl -n kube-system create serviceaccount kommander-cluster-admin
   ```

2. Create a token secret for the `serviceaccount`:

   ```
   kubectl -n kube-system create  -f - <<EOF
   apiVersion: v1
   kind: Secret
   metadata:
     name: kommander-cluster-admin-sa-token
     annotations:
       kubernetes.io/service-account.name: kommander-cluster-admin
   type: kubernetes.io/service-account-token
   EOF
   ```

   For more information on Service Account Tokens, refer to this article[369] in our blog.

3. Verify that the `serviceaccount` token is ready by running this command:

   ```
   kubectl -n kube-system get secret kommander-cluster-admin-sa-token -oyaml
   ```

   Verify that the `data.token` field is populated. The output should be similar to this:

   ```
   apiVersion: v1
   data:
     ca.crt: LS0tLS1CRUdJTiBDR...
     namespace: ZGVmYXVsdA==
     token: ZXlKaGJHY2lPaUpTVX...
   kind: Secret
   metadata:
     annotations:
       kubernetes.io/service-account.name: kommander-cluster-admin
       kubernetes.io/service-account.uid: b62bc32e-b502-4654-921d-94a742e273a8
     creationTimestamp: "2022-08-19T13:36:42Z"
     name: kommander-cluster-admin-sa-token
   ```

367 https://kubernetes.io/docs/tasks/tools/#kubectl
368 https://kubernetes.io/docs/concepts/cluster-administration/
369 https://eng.d2iq.com/blog/service-account-tokens-in-kubernetes-v1.24/#whats-changed-in-kubernetes-v124

```
  namespace: default
  resourceVersion: "8554"
  uid: 72c2a4f0-636d-4a70-9f1c-55a75f15e520
type: kubernetes.io/service-account-token
```

4. Configure the new service account for `cluster-admin` permissions:

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kommander-cluster-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kommander-cluster-admin
  namespace: kube-system
EOF
```

5. Set up the following environment variables with the access data that is needed for producing a new kubeconfig file:

```
export USER_TOKEN_VALUE=$(kubectl -n kube-system get secret/kommander-cluster-
admin-sa-token -o=go-template='{{.data.token}}' | base64 --decode)
export CURRENT_CONTEXT=$(kubectl config current-context)
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-
template='{{range .contexts}}{{if eq .name "'''${CURRENT_CONTEXT}'''"}}
{{ index .context "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}"{{with
index .cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{ end }}')
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}
{{ .cluster.server }}{{end}}{{ end }}')
```

6. Confirm these variables have been set correctly:

```
export -p | grep -E 'USER_TOKEN_VALUE|CURRENT_CONTEXT|CURRENT_CLUSTER|
CLUSTER_CA|CLUSTER_SERVER'
```

7. Generate a kubeconfig file that uses the environment variable values from the previous step:

```
cat << EOF > kommander-cluster-admin-config
```

```
apiVersion: v1
kind: Config
current-context: ${CURRENT_CONTEXT}
contexts:
- name: ${CURRENT_CONTEXT}
  context:
    cluster: ${CURRENT_CONTEXT}
    user: kommander-cluster-admin
    namespace: kube-system
clusters:
- name: ${CURRENT_CONTEXT}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
users:
- name: kommander-cluster-admin
  user:
    token: ${USER_TOKEN_VALUE}
EOF
```

8. This process produces a file in your current working directory called `kommander-cluster-admin-config`. The contents of this file are used in Kommander to attach the cluster.
   Before importing this configuration, verify the `kubeconfig` file can access the cluster:

   ```
   kubectl --kubeconfig $(pwd)/kommander-cluster-admin-config get all --all-namespaces
   ```

4.12.1.9.2 Attach from UI or Attach Manually Through the CLI

There are two options to attach the cluster. Choose from one of the following based on your preference.

- Through the UI (see page 438)
- Using the DKP CLI (see page 439)

4.12.1.9.2.1 Finish Attach EKS Cluster from the UI

Now that you have kubeconfig, go to the DKP UI and follow these steps below:

1. From the top menu bar, select your target workspace.
2. On the Dashboard page, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
3. Select **Attach Cluster**.
4. Select the **No additional networking restrictions** card. Alternatively, if you must use network restrictions, stop following the steps below, and see the instructions on the page Attach a cluster WITH network restrictions (see page 804).

5. Upload the kubeconfig file you created in the previous section (or copy its contents) into the **Cluster Configuration** section.
6. The **Cluster Name** field automatically populates with the name of the cluster in the kubeconfig. You can edit this field with the name you want for your cluster.
7. Add labels to classify your cluster as needed.
8. Select **Create** to attach your cluster.

If a cluster has limited resources to deploy all the federated platform services, it will fail to stay attached in the DKP UI. If this happens, ensure your system has sufficient resources for all pods.

4.12.1.9.3 Manually Attach a DKP CLI Cluster to the Management Cluster

These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

```
echo ${MANAGED_CLUSTER_NAME}
```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
{MANAGED_CLUSTER_NAME}.conf
```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
NOTE: This is only necessary if you never set the workspace of your cluster upon creation.
4. Retrieve the workspace where you want to attach the cluster:

```
kubectl get workspaces -A
```

5. Set the WORKSPACE_NAMESPACE environment variable:

```
export WORKSPACE_NAMESPACE=<workspace-namespace>
```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

```
kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
template='{{.data.value}}{{ "\n"}}'
```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
```

```
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

4.12.1.9.4 Related Information

For information on related topics or procedures, refer to the following:

- Create an EKS Cluster from the UI (see page 1278)
- Configuring and Running Amazon EKS Clusters[370]
- Manage Clusters (see page 774)

4.12.1.9.5 Next Step:

Day 2 - Cluster Operations Management (see page 590)

4.13 vSphere Install Options

For vSphere, an installation scenario is provided for you in this location.

Remember, there are always more options for custom YAML in the Custom Install and Additional Infrastructure Tools (see page 1050) in the vSphere Infrastructure (see page 1354) section, but this will get you operative in the most common scenarios.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

Below are all the supported environment variable combinations: (Refer to this page to check if your OS is a Supported Operating System (see page 67).)

- vSphere Prerequisites: All Installation Types (see page 442)
- vSphere Install (see page 447)
- vSphere Air-gapped Install (see page 468)
- vSphere FIPS Install (see page 492)
- vSphere FIPS Air-gapped Install (see page 514)

370 https://aws.amazon.com/eks/

4.13.1 Overview

vSphere is a more complex setup than some of the other providers and infrastructures, so an overview of steps has been provided to help.

The overall process for configuring vSphere and DKP together includes the following steps:

1. Configure vSphere to provide the needed elements, described in the vSphere Prerequisites: All Installation Types (see page 442).
2. **For air-gapped environments:** Create a bastion VM host (see page 1068).
3. Create a base OS image (for use in the OVA package containing the disk images packaged with the OVF).
4. Create a CAPI VM image template that uses the base OS image and adds the needed Kubernetes cluster components.
5. Create a new self-managing cluster on vSphere.
6. Install Kommander.
7. Verify and log in to the UI.

4.13.2 Next Step:

vSphere Prerequisites: All Installation Types (see page 442)

4.13.3 vSphere Prerequisites: All Installation Types

This section contains all the prerequisite information specific to VMware vSphere infrastructure. These are above and beyond all of the DKP prerequisites for Install.

4.13.3.1 vSphere: Minimum User Permissions

4.13.3.1.1 Create minimum required roles for provisioning and installing in vSphere

When a user needs permissions less than Admin, a role must be created with those permissions.

In small vSphere environments, with just a few hosts, assigning the role/user at the top level and propagating to child resources could be appropriate as shown on this page in the permissions tree below.

However, in the majority of cases this is not possible as security teams will enforce strict restrictions of who should have access to specific resources.

The process for configuring a vSphere role with the permissions for provisioning nodes and installing includes the following steps:

1. Open a vSphere Client connection to the vCenter Server, described in the Prerequisites (see page 1356).
2. Select Home > Administration > Roles > Add Role.
3. Give the new role a name, then select these Privileges:

Cns		
	☑	Searchable
Datastore		
	☑	Allocate space
	☑	Low level file operations
Host		
	• Configuration	
	☑	Storage partition configuration
Profile-driven storage		
	☑	Profile-driven storage view
Network		
	☑	Assign network
Resource		
	☑	Assign virtual machine to resource pool
Virtual machine		
	• Change Configuration - from the list in that section, select these permissions below:	

	☑	Add new disk
	☑	Add existing disk
	☑	Add or remove device
	☑	Advanced configuration
	☑	Change CPU count
	☑	Change Memory
	☑	Change Settings
	☑	Reload from path
	Edit inventory	
	☑	Create from existing
	☑	Remove
	Interaction	
	☑	Power off
	☑	Power on
	Provisioning	
	☑	Clone template
	☑	Deploy template
	Session	
	☑	ValidateSession

In the table below we describe the level at which these permissions should get assigned to.

Level	Required	Propagate to Child
vCenter Server (Top Level)	No	No
Data Center	Yes	No
Resource Pool	Yes	No
Folder	Yes	Yes
Template	Yes	No

4.13.3.1.2 Next Step:

vSphere: Storage Options (see page 445)

4.13.3.2 vSphere: Storage Options

4.13.3.2.1 Explore storage options and considerations for using DKP with VMware vSphere

The vSphere Container Storage[371] plugin supports shared NFS, vNFS, and vSAN. You need to provision your storage options in vCenter prior to creating a CAPI VM Template (see page 449) for non-air-gapped or in air-gapped (see page 472) in DKP for use with vSphere.

DKP has integrated the CSI 2.x driver[372] used in vSphere. When creating your DKP cluster, DKP uses whatever configuration you provide for the Datastore name. vSAN is not required. Using NFS can reduce the amount of tagging and permission granting required to configure your cluster.

4.13.3.2.2 Next Step:

vSphere: VMware Prerequisites (see page 446)

371 https://docs.vmware.com/en/VMware-vSphere-Container-Storage-Plug-in/2.0/vmware-vsphere-csp-getting-started/GUID-74AF02D7-1562-48BD-A9FE-C81A53342AC3.html
372 https://github.com/kubernetes-sigs/vsphere-csi-driver

4.13.3.3 vSphere: VMware Prerequisites

Before installing, verify that your VMware vSphere Client environment[373] meets the following basic requirements:

- Access to a bastion (see page 1068) VM, or other network-connected host, running vSphere Client version v6.7.x with Update 3 or later version.
    - You must be able to reach the vSphere API endpoint from where the Konvoy command line interface (CLI) runs.
- vSphere account with credentials configured - this account must have Administrator privileges (see page 442).
- A RedHat® subscription with username and password for downloading DVD ISOs.
- Valid vSphere values for the following:
    - vCenter API server URL
    - Datacenter name
    - Zone name that contains ESXi hosts[374] for your cluster's nodes.
    - Datastore name for the shared storage resource to be used for the VMs in the cluster.
        - Use of PersistentVolumes in your cluster depends on Cloud Native Storage (CNS), available in vSphere v6.7.x with Update 3 and later versions. CNS depends on this shared Datastore's configuration.
    - Datastore URL from the datastore record for the shared datastore you want your cluster to use.
        - You need this URL value to ensure that the correct Datastore is used when DKP creates VMs for your cluster in vSphere.
    - Folder name
    - Base template name, such as base-rhel-8, or base-rhel-7.
    - Name of a Virtual Network that has DHCP enabled for both air-gapped and non air-gapped environments.
    - Name of your virtual IP interface.
    - Resource Pools - at least one resource pool is needed, with every host in the pool having access to shared storage, such as VSAN.
        - Each host in the resource pool needs access to shared storage, such as NFS or VSAN, to make use of MachineDeployments and high-availability control planes.

4.13.3.3.1 Next Step:

vSphere Install (see page 447)

---

373 https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.vsphere.vm_admin.doc/GUID-55238059-912E-411F-A0E9-A7A536972A91.html

374 https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.esxi.install.doc/GUID-B2F01BF5-078A-4C7E-B505-5DFFED0B8C38.html

4.13.4 vSphere Install

This section provides instructions to install DKP in a vSphere non-air-gapped environment.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

4.13.4.1 Further vSphere Prerequisites

- vSphere: VMware Prerequisites (see page 446)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430985/vSphere+Prerequisites+-+Storage+Options (see page 445)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430977/vSphere+Prerequisites+-+Minimum+User+Permissions (see page 442)

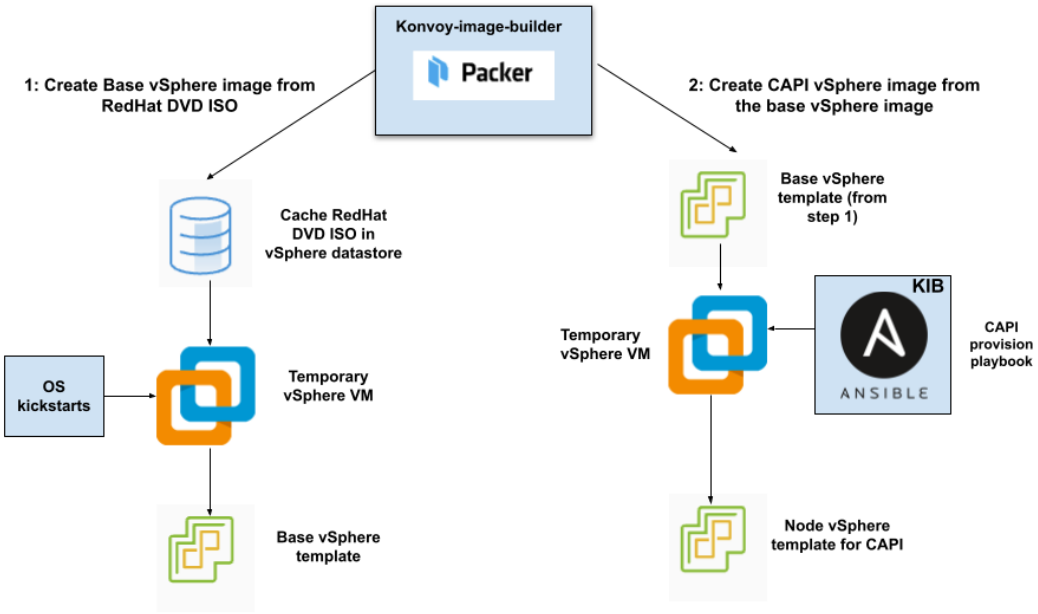
4.13.4.2 Next Step:

vSphere: Image Creation Overview (see page 447)

4.13.4.3 vSphere: Image Creation Overview

4.13.4.3.1 Image Creation Process

This diagram illustrates the image creation process:



The workflow on the left shows the creation of a base OS image in the vCenter vSphere client using inputs from Packer. The workflow on the right shows how DKP uses that same base OS image to create CAPI-enabled VM images for your cluster.

After creating the base image, the DKP image builder uses it to create a CAPI-enabled vSphere template that includes the Kubernetes objects for the cluster. You can use that resulting template with the DKP `create cluster` command to create the VM nodes in your cluster directly on a vCenter server. From that point, you can use DKP to provision and manage your cluster.

4.13.4.3.2 Next Step:

vSphere: Create an Image (see page 448)

4.13.4.4 vSphere: Create an Image

Creating a base OS image from DVD ISO files is a one-time process. The base OS image file is created in the vSphere Client for use in the vSphere VM template. Therefore, the base OS image is used by Konvoy Image Builder (see page 1626)(KIB) to create a VM template to configure Kubernetes nodes by the DKP vSphere provider.

4.13.4.4.1 Create the Base OS Image

For vSphere, a username is populated by `SSH_USERNAME` and the user can use authorization via `SSH_PASSWORD` or `SSH_PRIVATE_KEY_FILE` environment variables and required by default by packer. This user should have administrator privileges. It is possible to configure a custom user and password when building the OS image, however, that requires the Konvoy Image Builder (KIB) configuration to be overridden (see page 1670).

While creating the base OS image, it is important to take into consideration the following elements:

- Storage configuration: D2iQ recommends customizing disk partitions and not configuring a SWAP partition.
- Network configuration: as KIB must download and install packages, activating the network is required.
- Connect to Red Hat: if using RHEL, registering with Red Hat is required to configure software repositories and install software packages.
- Software selection: D2iQ recommends choosing **Minimal Install**.
- DKP recommends to install with the packages provided by the operating system package managers. Use the version that corresponds to the major version of your operating system.

vSphere: Create a CAPI VM Template Image (see page 449)

4.13.4.5 vSphere: Create a CAPI VM Template

4.13.4.5.1 Prerequisites

- Users need to vSphere: Create an Image (see page 448) before starting this procedure.
- Build image template with Konvoy Image Builder (KIB)

- If using GPU, prepare an OS image for your NVIDIA GPU nodepool, using the Konvoy Image Builder instructions here: KIB for GPU (see page 1649).

4.13.4.5.2 Create a vSphere Template for Your Cluster from a Base OS Image

Using the base OS image created in a previous procedure, DKP creates the new vSphere template directly on the vCenter server.

1. Set the following vSphere environment variables on the bastion VM host:

```
export VSPHERE_SERVER=your_vCenter_APIserver_URL
export VSPHERE_USERNAME=your_vCenter_user_name
export VSPHERE_PASSWORD=your_vCenter_password
```

2. Copy the base OS image file created in the vSphere Client to your desired location on the bastion VM host and make a note of the path and file name.
3. Create an `image.yaml` file and add the following variables for vSphere. DKP uses this file and these variables as inputs in the next step. To customize your `image.yaml` file, refer to this section: Customize your Image (see page 1661).
   ⚠️ **NOTE**: This example is Ubuntu 20.04. You will need to replace OS name below based on your OS. See other default YAML examples[375] for copy and paste below last step.

```
---
download_images: true
build_name: "ubuntu-2004"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-
init-vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: "<VSPHERE_CLUSTER_NAME>"
  datacenter: "<VSPHERE_DATACENTER_NAME>"
  datastore: "<VSPHERE_DATASTORE_NAME>"
  folder: "<VSPHERE_FOLDER>"
  insecure_connection: "false"
  network: "<VSPHERE_NETWORK>"
  resource_pool: "<VSPHERE_RESOURCE_POOL>"
  template: "os-qualification-templates/d2iq-base-Ubuntu-20.04" # change
default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "ubuntu2004-64"
  # goss params
  distribution: "ubuntu"
  distribution_version: "20.04"
# Use following overrides to select the authentication method that can be used
with base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key
will be ignored
```

375 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

4. Create a vSphere VM template with your variation of the following command:

```
konvoy-image build images/ova/<image.yaml>
```

Any additional configurations can be added to this command using `--overrides` flags as shown below:

   a. Any credential overrides: `--overrides overrides.yaml`
   b. for FIPS, add this flag: `--overrides overrides/fips.yaml`
   c. for air-gapped, add this flag: `--overrides overrides/offline-fips.yaml`

5. The Konvoy Image Builder (see page 1652) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. This template contains the required artifacts needed to create a Kubernetes cluster.
When KIB provisions the OS image[376] successfully, it creates a manifest file. The `artifact_id` field of this file contains the name of the AMI ID (AWS), template name (vSphere), or image name (GCP/Azure), for example:


```
{
      "name": "vsphere-clone",
      "builder_type": "vsphere-clone",
      "build_time": 1644985039,
      "files": null,
      "artifact_id": "konvoy-ova-vsphere-rhel-84-1.21.6-1644983717",
      "packer_run_uuid": "260e8110-77f8-ca94-e29e-ac7a2ae779c8",
      "custom_data": {
        "build_date": "2022-02-16T03:55:17Z",
        "build_name": "vsphere-rhel-84",
        "build_timestamp": "1644983717",
        [...]
      }
    }
```


Recommendation: Now we can now see the template created in our vCenter, it is best to rename it to `dkp-<DKP_VERSION>-k8s-<K8S_VERSION>-<DISTRO>`, like `dkp-2.4.0-k8s-1.24.6-ubuntu` to keep templates organized.

6. Next steps are to deploy a DKP cluster using your vSphere template.

4.13.4.5.2.1 Additional OS YAML files for copy/paste:

RHEL 8.6

376 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

```
---
download_images: true
build_name: "rhel-86"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "os-qualification-templates/d2iq-base-RHEL-86" # change default value
with your base template name
  vsphere_guest_os_type: "rhel8_64Guest"
  guest_os_type: "rhel8-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "8.6"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Rocky Linux 9.1

```
---
download_images: true
build_name: "rocky-91"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
```

```
  network: ""
  resource_pool: ""
  template: "os-qualification-templates/d2iq-base-RockyLinux-9.1" # change default
 value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "rocky9-64"
  # goss params
  distribution: "rocky"
  distribution_version: "9.1"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

4.13.4.5.2.2 Next Step:

vSphere: Create the Management Cluster (see page 453)

4.13.4.6 vSphere: Create the Management Cluster

Use this procedure to create a self-managed vSphere Management cluster with DKP. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

4.13.4.6.1 Name your Cluster

1. Give your cluster a unique name suitable for your environment. The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[377] for more naming information.
2. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=<my-vsphere-cluster>
```

377 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

4.13.4.6.2 Create a New vSphere Kubernetes Cluster

DKP uses the vsphere CSI driver as the default storage provider (see page 110). Use a Kubernetes CSI[378] compatible storage that is suitable for production.

The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.
If you do not set a workspace, it will be created in the `default` workspace, and you need to take additional steps to attach to a workspace later. For instructions on how to do this, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster). (see page 784).

Follow these steps to create the cluster:

1. Use the following command to set the environment variables for vSphere:

```
export VSPHERE_SERVER=example.vsphere.url
export VSPHERE_USERNAME=user@example.vsphere.url
export VSPHERE_PASSWORD=example_password
```

2. Generate the Kubernetes cluster objects by copying and editing this command to include the correct values, including the VM template name you assigned in the previous procedure:

- To increase Dockerhub's rate limit[379] use your Dockerhub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.
- The following example shows a common configuration. See dkp create cluster (see page 1853) reference for the full list of cluster creation options:

```
dkp create cluster vsphere \
```

378 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
379 https://docs.docker.com/docker-hub/download-rate-limit/

```
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <xxx.yyy.zzz.000> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file <SSH_PUBLIC_KEY_FILE> \
  --resource-pool <RESOURCE_POOL_NAME> \
  --vm-template <TEMPLATE_NAME> \
  --virtual-ip-interface <ip_interface_name> \
  --self-managed
```

Flatcar OS

Flatcar OS (see page 1183) use `--os-hint` to instruct the bootstrap cluster to make some changes related to the installation paths:

```
--os-hint flatcar
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

For bootstrap and custom YAML cluster creation, refer to the Custom Installation and Additional Infrastructure Tools (see page 1050) section of the documentation for vSphere: vSphere Infrastructure (see page 1354)

4.13.4.6.2.1 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.13.4.6.2.2 Known Limitations

Be aware of these limitations in the current release of DKP Konvoy.

- The DKP Konvoy version used to create a bootstrap cluster must match the DKP Konvoy version used to create a workload cluster.
- DKP Konvoy supports deploying one workload cluster.
- DKP Konvoy generates a set of objects for one Node Pool.
- DKP Konvoy does not validate edits to cluster objects.

4.13.4.6.2.3 Next Step:

vSphere: Configure Metal LB (see page 456)

4.13.4.7 vSphere: Configure Metal LB

If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can proceed to vSphere: Install Kommander (see page 458) and continue the installation process.

To use MetalLB, create a MetalLB configMap for your vSphere infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

4.13.4.7.1 Layer 2 Configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

- MetalLB IP address ranges/CIDRs should be within the node's primary network subnet (see page 1065).
- MetalLB IP address ranges/CIDRs and node subnet should not conflict with the Kubernetes cluster pod and service subnets.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

When complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.13.4.7.2 BGP configuration

BGP, like any routing protocol, requires a mesh, so you configure BGP between the nodes. If you are in a Data Center, peer with the BGP routers. The instructions for use with Cloud providers is the same. For a standard configuration featuring one BGP router and one IP address range, you need four pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

Extract the kubeconfig and deploy a config map for MetalLB using the following command:

```
dkp get kubeconfig -c ${DKP_CLUSTER_NAME} > ${DKP_CLUSTER_NAME}.conf
```

Deploy MetalLB Configuration with the command below:

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

When complete, run the following `kubectl` command:

```
kubectl apply -f metallb-conf.yaml
```

4.13.4.7.3 Next Step:

vSphere: Install Kommander (see page 458)

4.13.4.8 vSphere: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

4.13.4.8.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

4.13.4.8.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1558) for the deployment:

```
dkp install kommander --init > kommander.yaml
```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1541)), etc.

4.13.4.8.1.2 Enable DKP Catalog Applications and Install Kommander

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.7.0
```

   If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.13.4.8.1.3 Next Step:

vSphere: Verify Install and Log in to the UI (see page 460)

4.13.4.9 vSphere: Verify Install and Log in to the UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.13.4.9.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

4.13.4.9.2 Log in to the UI with Kommander

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

4.13.4.9.2.1 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

4.13.4.9.2.2 Next Steps:

vSphere: Create a Managed Cluster Using the DKP CLI (see page 463)

4.13.4.10 vSphere: Create a Managed Cluster Using the DKP CLI

Enterprise Gov Advanced

Creating a vSphere Managed cluster with the DKP CLI assumes that you already fulfilled all of the prerequisites and successfully created a vSphere Management cluster. Use this procedure to create a Managed vSphere cluster.

> ⚠ When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

4.13.4.10.1 Make New Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.13.4.10.2 Name your Cluster

1. Give your cluster a unique name suitable for your environment. The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[380] for more naming information.
2. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=<my-managed-vsphere-cluster>
```

380 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

4.13.4.10.3 Create a New vSphere Kubernetes Cluster

DKP uses the vsphere CSI driver as the default storage provider (see page 110). Use a Kubernetes CSI[381] compatible storage that is suitable for production.

The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.
If you do not set a workspace, it will be created in the `default` workspace, and you need to take additional steps to attach to a workspace later. For instructions on how to do this, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster). (see page 784).

Follow these steps to create the cluster:

1. Use the following command to set the environment variables for vSphere:

```
export VSPHERE_SERVER=example.vsphere.url
export VSPHERE_USERNAME=user@example.vsphere.url
export VSPHERE_PASSWORD=example_password
```

2. Generate the Kubernetes cluster objects by copying and editing this command to include the correct values, including the VM template name you assigned in the previous procedure:

- To increase Dockerhub's rate limit[382] use your Dockerhub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.
- The following example shows a common configuration. See dkp create cluster (see page 1853) reference for the full list of cluster creation options:

```
dkp create cluster vsphere \
```

381 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
382 https://docs.docker.com/docker-hub/download-rate-limit/

```
--cluster-name ${MANAGED_CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--namespace ${WORKSPACE_NAMESPACE} \
--network <NETWORK_NAME> \
--control-plane-endpoint-host <xxx.yyy.zzz.000> \
--data-center <DATACENTER_NAME> \
--data-store <DATASTORE_NAME> \
--folder <FOLDER_NAME> \
--server <VCENTER_API_SERVER_URL> \
--ssh-public-key-file <SSH_PUBLIC_KEY_FILE> \
--resource-pool <RESOURCE_POOL_NAME> \
--virtual-ip-interface <ip_interface_name> \
--vm-template <TEMPLATE_NAME> \
--kubeconfig=<management-cluster-kubeconfig-path> \
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

4.13.4.10.4 Retrieve the kubeconfig and Explore New vSphere Cluster

Follow these steps:

1. Fetch the kubeconfig file with the command:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} --kubeconfig
   <management-cluster-kubeconfig-path> -n ${WORKSPACE_NAMESPACE} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

2. List the nodes with the following command:

   ```
   kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get nodes
   ```

3. List the pods with the following command:

NOTE: Wait for the Status to move to Ready while `calico-node` pods are being deployed.

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get pods -A
```

4.13.4.10.4.1 Manually Attach a DKP CLI Cluster to the Management Cluster

> ⚠ These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

   ```
   echo ${MANAGED_CLUSTER_NAME}
   ```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
   NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

   ```
   kubectl get workspaces -A
   ```

5. Set the WORKSPACE_NAMESPACE environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace-namespace>
   ```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

   ```
   kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
   template='{{.data.value}}{{ "\n"}}'
   ```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

If you have existing clusters or want to create other new clusters to attach, there are many ways to attach a cluster with various requirements and restrictions. To see all the options, visit the section in documentation Day 2 - Attach an Existing Kubernetes Cluster (see page 784).

At this point, you can create more clusters, perform other configuration tasks (see page 1050), or proceed to Day 2 Operations (see page 590).

4.13.5 vSphere Air-gapped Install

This section provides instructions to install DKP in a vSphere **air-gapped** environment.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

> For air-gapped, ensure you have downloaded (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , so you can extract the tarball on the page with those instructions.

4.13.5.1 Further vSphere Prerequisites

- vSphere: VMware Prerequisites (see page 446)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430985/vSphere+Prerequisites+-+Storage+Options (see page 445)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430977/vSphere+Prerequisites+-+Minimum+User+Permissions (see page 442)

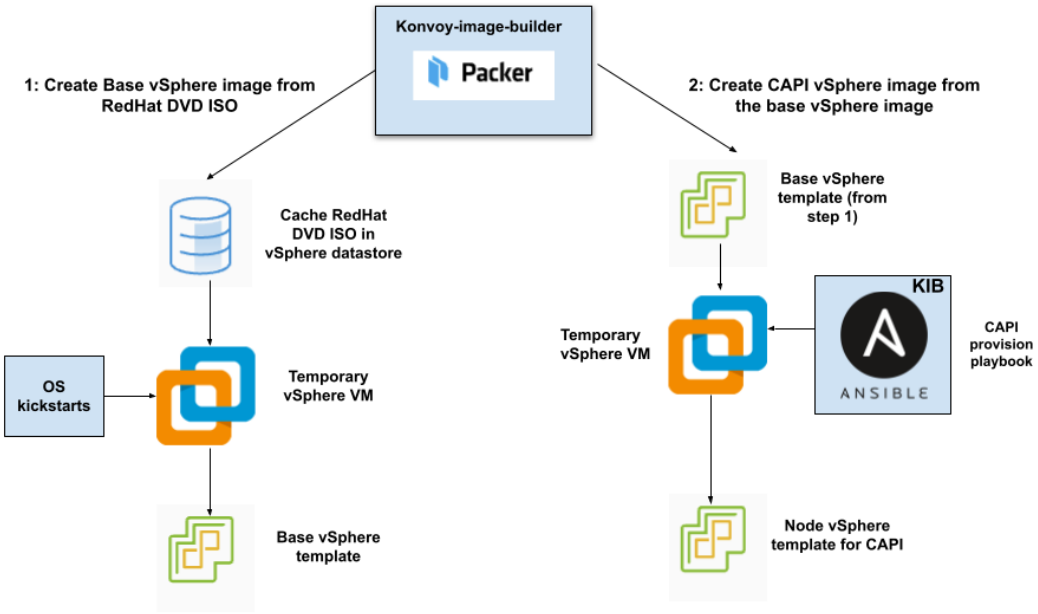
4.13.5.2 Next Step:

vSphere Air-gapped: Image Creation Overview (see page 468)

4.13.5.3 vSphere Air-gapped: Image Creation Overview

4.13.5.3.1 Image Creation Process

This diagram illustrates the image creation process:



The workflow on the left shows the creation of a base OS image in the vCenter vSphere client using inputs from Packer. The workflow on the right shows how DKP uses that same base OS image to create CAPI-enabled VM images for your cluster.

After creating the base image, the DKP image builder uses it to create a CAPI-enabled vSphere template that includes the Kubernetes objects for the cluster. You can use that resulting template with the DKP `create cluster` command to create the VM nodes in your cluster directly on a vCenter server. From that point, you can use DKP to provision and manage your cluster.

4.13.5.3.2 Next Step:

vSphere Air-gapped: Create an Image (see page 469)

4.13.5.4 vSphere Air-gapped: Create an Image

Creating a base OS image from DVD ISO files is a one-time process. The base OS image file is created in the vSphere Client for use in the vSphere VM template. Therefore, the base OS image is used by Konvoy Image Builder (see page 1626)(KIB) to create a VM template to configure Kubernetes nodes by the DKP vSphere provider.

4.13.5.4.1 Create the Base OS Image

For vSphere, a username is populated by `SSH_USERNAME` and the user can use authorization via `SSH_PASSWORD` or `SSH_PRIVATE_KEY_FILE` environment variables and required by default by packer. This user should have administrator privileges. It is possible to configure a custom user and password when building the OS image, however, that requires the Konvoy Image Builder (KIB) configuration to be overridden (see page 1670).

While creating the base OS image, it is important to take into consideration the following elements:

- Storage configuration: D2iQ recommends customizing disk partitions and not configuring a SWAP partition.
- Network configuration: as KIB must download and install packages, activating the network is required.
- Connect to Red Hat: if using RHEL, registering with Red Hat is required to configure software repositories and install software packages.
- Software selection: D2iQ recommends choosing **Minimal Install**.
- DKP recommends to install with the packages provided by the operating system package managers. Use the version that corresponds to the major version of your operating system.

4.13.5.4.2 Next Step:

vSphere Air-gapped: Docker Registry (see page 470)

4.13.5.5 vSphere Air-gapped: Load the Registry

Before creating an air-gapped Kubernetes cluster, you need to load the required images in a local registry for the Konvoy component. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1606) page for more information.

1. If not already done in prerequisites, download (see page 76) the air-gapped bundle `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap cluster, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

```
cd dkp-v2.8.1
```

3. Set an environment variable with your registry address and any other needed variables using this command:

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
export REGISTRY_CA=<path to the cacert file on the bastion>
```

4. Execute the following command to load the air-gapped image bundle into your private registry using any of the relevant flags to apply variables above:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --
to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-
registry-password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

4.13.5.5.1 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

4.13.5.5.2 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

4.13.5.5.3 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME}
--to-registry-password=${REGISTRY_PASSWORD}
```

4.13.5.5.4 Next Step:

vSphere Air-gapped: Create a CAPI VM Template Image (see page 472)

4.13.5.6 vSphere Air-gapped: Create a CAPI VM Template

You must have at least one image before creating a new cluster. As long as you have an image, this step in your configuration is not required each time since that image can be used to spin up a new cluster. However, if you need different images for different environments or providers, you will need to create a new custom image.

Using KIB (see page 1626), you can create your VM template without requiring access to the internet by providing an additional `--override` flag.

1. Assuming you have downloaded (see page 76) `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz`, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz && cd dkp-v2.8.0/kib
   ```

2. Follow the instructions to build a vSphere template below and set the override `--overrides overrides/offline.yaml` flag.

4.13.5.6.1 Create a vSphere Template for Your Cluster from a Base OS Image

Using the base OS image created in a previous procedure, DKP creates the new vSphere template directly on the vCenter server.

1. Set the following vSphere environment variables on the bastion VM host:

   ```
   export VSPHERE_SERVER=your_vCenter_APIserver_URL
   ```

```
export VSPHERE_USERNAME=your_vCenter_user_name
export VSPHERE_PASSWORD=your_vCenter_password
```

2. Copy the base OS image file created in the vSphere Client to your desired location on the bastion VM host and make a note of the path and file name.
3. Create an `image.yaml` file and add the following variables for vSphere. DKP uses this file and these variables as inputs in the next step. To customize your `image.yaml` file, refer to this section: Customize your Image (see page 1661).
   ⚠ **NOTE**: This example is Ubuntu 20.04. You will need to replace OS name below based on your OS. See other default YAML examples[383] for copy and paste below last step.

```
---
download_images: true
build_name: "ubuntu-2004"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-
init-vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: "<VSPHERE_CLUSTER_NAME>"
  datacenter: "<VSPHERE_DATACENTER_NAME>"
  datastore: "<VSPHERE_DATASTORE_NAME>"
  folder: "<VSPHERE_FOLDER>"
  insecure_connection: "false"
  network: "<VSPHERE_NETWORK>"
  resource_pool: "<VSPHERE_RESOURCE_POOL>"
  template: "os-qualification-templates/d2iq-base-Ubuntu-20.04" # change
default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "ubuntu2004-64"
  # goss params
  distribution: "ubuntu"
  distribution_version: "20.04"
# Use following overrides to select the authentication method that can be used
with base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key
will be ignored
```

383 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

4. Create a vSphere VM template with your variation of the following command:

   ```
   konvoy-image build images/ova/<image.yaml>
   ```

   Any additional configurations can be added to this command using `--overrides` flags as shown below:

   a. Any credential overrides: `--overrides overrides.yaml`
   b. for FIPS, add this flag: `--overrides overrides/fips.yaml`
   c. for air-gapped, add this flag: `--overrides overrides/offline-fips.yaml`

5. The Konvoy Image Builder (see page 1652) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. This template contains the required artifacts needed to create a Kubernetes cluster.
   When KIB provisions the OS image[384] successfully, it creates a manifest file. The `artifact_id` field of this file contains the name of the AMI ID (AWS), template name (vSphere), or image name (GCP/Azure), for example:

   ```
   {
         "name": "vsphere-clone",
         "builder_type": "vsphere-clone",
         "build_time": 1644985039,
         "files": null,
         "artifact_id": "konvoy-ova-vsphere-rhel-84-1.21.6-1644983717",
         "packer_run_uuid": "260e8110-77f8-ca94-e29e-ac7a2ae779c8",
         "custom_data": {
           "build_date": "2022-02-16T03:55:17Z",
           "build_name": "vsphere-rhel-84",
           "build_timestamp": "1644983717",
           [...]
         }
       }
   ```

   Recommendation: Now we can now see the template created in our vCenter, it is best to rename it to `dkp-<DKP_VERSION>-k8s-<K8S_VERSION>-<DISTRO>`, like `dkp-2.4.0-k8s-1.24.6-ubuntu` to keep templates organized.

6. Next steps are to deploy a DKP cluster using your vSphere template.

4.13.5.6.1.1 Additional OS YAML files for copy/paste:

RHEL 8.6

384 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

```
---
download_images: true
build_name: "rhel-86"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rhel-8.6" # change default value with your base template name
  vsphere_guest_os_type: "rhel8_64Guest"
  guest_os_type: "rhel8-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "8.6"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Ubuntu 20.04

```
---
download_images: true
build_name: "ubuntu-2004"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
```

```
  resource_pool: ""
  template: "base-ubuntu-20.04" # change default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "ubuntu2004-64"
  # goss params
  distribution: "ubuntu"
  distribution_version: "20.04"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Rocky Linux 9.1

```
---
download_images: true
build_name: "rocky-91"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rocky-9.1" # change default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "rocky9-64"
  # goss params
  distribution: "rocky"
  distribution_version: "9.1"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

4.13.5.6.1.2 Next Step:

vSphere Air-gapped: Create the Management Cluster (see page 477)

4.13.5.7 vSphere Air-gapped: Create the Management Cluster

This page of instructions will create a self-managed air-gapped management cluster. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

4.13.5.7.1 Name your Cluster

1. Give your cluster a unique name suitable for your environment. The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[385] for more naming information.
2. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=<my-vsphere-cluster>
```

DKP uses the vsphere CSI driver as the default storage provider (see page 110). Use a Kubernetes CSI[386] compatible storage that is suitable for production.

4.13.5.7.1.1 Create a New vSphere Kubernetes Cluster

The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.
If you do not set a workspace, it will be created in the `default` workspace, and you need to take additional steps to attach to a workspace later. For instructions on how to do this, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster). (see page 784).

Use the following steps to create a new, air-gapped vSphere cluster.

385 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/
386 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

1. Configure your cluster to use an existing local registry as a mirror when attempting to pull images: **IMPORTANT:** The image must be created by Konvoy Image Builder (see page 1652) in order to use the registry mirror feature.

   ```
   export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
   export REGISTRY_CA=<path to the CA on the bastion>
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   ```

   - `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
   - `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
   - `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
   - `REGISTRY_PASSWORD` : optional if username is not set.

2. Load the image, using either the `docker` or `podman` command:

   ```
   docker load -i konvoy-bootstrap-image-v2.8.0.tar
   ```

   OR

   ```
   podman load -i konvoy-bootstrap-image-v2.8.0.tar
   ```

3. Create a Kubernetes cluster by copying the following command and substituting the valid values for your environment:

```
dkp create cluster vsphere \
  --cluster-name=${MANAGED_CLUSTER_NAME} \
  --additional-tags=owner=$(whoami) \
  --kubeconfig=<management-cluster-kubeconfig-path> \
  --namespace ${WORKSPACE_NAMESPACE}
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <CONTROL_PLANE_IP> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file </path/to/key.pub> \
  --resource-pool <RESOURCE_POOL_NAME> \
  --vm-template konvoy-ova-vsphere-os-release-k8s_release-vsphere-timestamp \
```

```
  --virtual-ip-interface <ip_interface_name> \
  --extra-sans "127.0.0.1" \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --self-managed
```

Flatcar OS

Flatcar OS (see page 1183) use `--os-hint` to instruct the bootstrap cluster to make some changes related to the installation paths:

```
--os-hint flatcar
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

For bootstrap and custom YAML cluster creation, refer to the Custom Installation and Additional Infrastructure Tools (see page 1050) section of the documentation for vSphere: vSphere Infrastructure (see page 1354)

4.13.5.7.2 Retrieve the kubeconfig and Explore New vSphere Cluster

Follow these steps:

1. Fetch the kubeconfig file with the command:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} --kubeconfig
   <management-cluster-kubeconfig-path> -n ${WORKSPACE_NAMESPACE} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

2. List the nodes with the following command:

   ```
   kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get nodes
   ```

3. List the pods with the following command:

NOTE: Wait for the Status to move to Ready while `calico-node` pods are being deployed.

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get pods -A
```

4.13.5.7.2.1 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

As they progress, the controllers create Events, which you can also monitor using the command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, this example uses `grep`. You can also use separate commands to get Events for specific objects, such as `kubectl get events --field-selector involvedObject.kind="VSphereCluster"` and `kubectl get events --field-selector involvedObject.kind="VSphereMachine"`.

4.13.5.7.2.2 Next Step:

vSphere Air-gapped: Configure Metal LB (see page 480)

4.13.5.8 vSphere Air-gapped: Configure Metal LB

If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can proceed to vSphere Air-gapped: Install Kommander (see page 483) and continue the installation process.

To use MetalLB, create a MetalLB configMap for your vSphere infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

4.13.5.8.1 Layer 2 Configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

- MetalLB IP address ranges/CIDRs should be within the node's primary network subnet (see page 1065).
- MetalLB IP address ranges/CIDRs and node subnet should not conflict with the Kubernetes cluster pod and service subnets.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

When complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.13.5.8.2 BGP configuration

BGP, like any routing protocol, requires a mesh, so you configure BGP between the nodes. If you are in a Data Center, peer with the BGP routers. The instructions for use with Cloud providers is the same. For a standard configuration featuring one BGP router and one IP address range, you need four pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,

- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

Extract the kubeconfig and deploy a config map for MetalLB using the following command:

```
dkp get kubeconfig -c ${DKP_CLUSTER_NAME} > ${DKP_CLUSTER_NAME}.conf
```

Deploy MetalLB Configuration with the command below:

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

When complete, run the following `kubectl` command:

```
kubectl apply -f metallb-conf.yaml
```

4.13.5.8.3 Next Step:

vSphere Air-gapped: Install Kommander (see page 483)

4.13.5.9 vSphere Air-gapped: Install Kommander

4.13.5.9.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry: Air-gapped Environments (see page 1536).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

4.13.5.9.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init --airgapped > kommander.yaml
   ```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1541)), etc.

4.13.5.9.1.2 Enable DKP Catalog Applications and Install Kommander in an Air-gapped Environment

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add the following values to enable DKP Catalog Applications:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./dkp-catalog-applications-v2.8.1.tar.gz
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.1.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.1.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.8.1.tar.gz
```

4.13.5.9.1.3 Next Step:

vSphere Air-gapped: Verify Install and Log in to the UI (see page 484)

4.13.5.10 vSphere Air-gapped: Verify Install and Log in to the UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

> **NOTE**: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.13.5.10.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a "broken" release state, such as "exhausted" or "another rollback/release in progress", trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

4.13.5.10.2 Log in to the UI with Kommander

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

4.13.5.10.2.1 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as

attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

4.13.5.10.2.2 Next Steps:

vSphere Air-gapped: Create a Managed Cluster Using the DKP CLI (see page 487)

4.13.5.11 vSphere Air-gapped: Create a Managed Cluster Using the DKP CLI

Creating an air-gapped vSphere Managed cluster with the DKP CLI assumes that you already fulfilled all of the prerequisites and successfully created a vSphere Management cluster. Use this procedure to create a Managed vSphere cluster.

When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

4.13.5.11.1 Make New Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.13.5.11.2 Name your Cluster

1. Give your cluster a unique name suitable for your environment. The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[387] for more naming information.
2. Set the CLUSTER_NAME environment variable with the command:

387 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
export CLUSTER_NAME=<my-managed-vsphere-cluster>
```

DKP uses the vsphere CSI driver as the default storage provider (see page 110). Use a Kubernetes CSI[388] compatible storage that is suitable for production.

4.13.5.11.2.1 Create a New vSphere Kubernetes Cluster

The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.
If you do not set a workspace, it will be created in the `default` workspace, and you need to take additional steps to attach to a workspace later. For instructions on how to do this, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster). (see page 784).

Use the following steps to create a new, air-gapped vSphere cluster.

1. Configure your cluster to use an existing local registry as a mirror when attempting to pull images: **IMPORTANT:** The image must be created by Konvoy Image Builder (see page 1652) in order to use the registry mirror feature.

   ```
   export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
   export REGISTRY_CA=<path to the CA on the bastion>
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   ```

   - `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
   - `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
   - `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
   - `REGISTRY_PASSWORD` : optional if username is not set.

2. Load the image, using either the `docker` or `podman` command:

388 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

```
docker load -i konvoy-bootstrap-image-v2.8.0.tar
```

OR

```
podman load -i konvoy-bootstrap-image-v2.8.0.tar
```

3. Create a Kubernetes cluster by copying the following command and substituting the valid values for your environment:

```
dkp create cluster vsphere \
  --cluster-name=${MANAGED_CLUSTER_NAME} \
  --additional-tags=owner=$(whoami) \
  --kubeconfig=<management-cluster-kubeconfig-path> \
  --namespace ${WORKSPACE_NAMESPACE}
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <CONTROL_PLANE_IP> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file </path/to/key.pub> \
  --resource-pool <RESOURCE_POOL_NAME> \
  --vm-template konvoy-ova-vsphere-os-release-k8s_release-vsphere-timestamp \
  --virtual-ip-interface <ip_interface_name> \
  --extra-sans "127.0.0.1" \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --kubeconfig=<management-cluster-kubeconfig-path> \
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

4.13.5.11.3 Retrieve the kubeconfig and Explore New vSphere Cluster

Follow these steps:

1. Fetch the kubeconfig file with the command:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} --kubeconfig
<management-cluster-kubeconfig-path> -n ${WORKSPACE_NAMESPACE} > $
{MANAGED_CLUSTER_NAME}.conf
```

2. List the nodes with the following command:

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get nodes
```

3. List the pods with the following command:

NOTE: Wait for the Status to move to Ready while `calico-node` pods are being deployed.

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get pods -A
```

4.13.5.11.3.1 Manually Attach a DKP CLI Cluster to the Management Cluster

These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

```
echo ${MANAGED_CLUSTER_NAME}
```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
{MANAGED_CLUSTER_NAME}.conf
```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
   NOTE: This is only necessary if you never set the workspace of your cluster upon creation.
4. Retrieve the workspace where you want to attach the cluster:

```
kubectl get workspaces -A
```

5. Set the WORKSPACE_NAMESPACE environment variable:

```
export WORKSPACE_NAMESPACE=<workspace-namespace>
```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

```
kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
template='{{.data.value}}{{ "\n"}}'
```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
```

```
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

If you have existing clusters or want to create other new clusters to attach, there are many ways to attach a cluster with various requirements and restrictions. To see all the options, visit the section in documentation Day 2 - Attach an Existing Kubernetes Cluster (see page 784).

At this point, you can create more clusters, perform other configuration tasks (see page 1050), or proceed to Day 2 Operations (see page 590).

4.13.6 vSphere FIPS Install

This section provides instructions to install DKP in a vSphere with FIPS.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

4.13.6.1 Further vSphere Prerequisites

- vSphere: VMware Prerequisites (see page 446)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430985/vSphere+Prerequisites+-+Storage+Options (see page 445)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430977/vSphere+Prerequisites+-+Minimum+User+Permissions (see page 442)

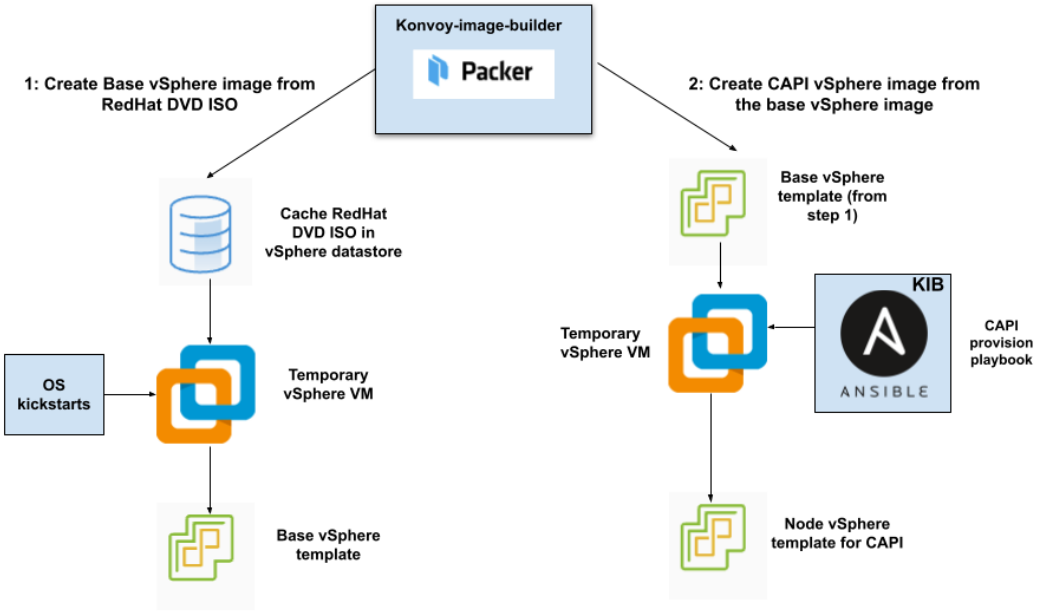
4.13.6.2 Next Step:

vSphere FIPS: Image Creation Overview (see page 493)

4.13.6.3 vSphere FIPS: Image Creation Overview

4.13.6.3.1 Image Creation Process

This diagram illustrates the image creation process:



The workflow on the left shows the creation of a base OS image in the vCenter vSphere client using inputs from Packer. The workflow on the right shows how DKP uses that same base OS image to create CAPI-enabled VM images for your cluster.

After creating the base image, the DKP image builder uses it to create a CAPI-enabled vSphere template that includes the Kubernetes objects for the cluster. You can use that resulting template with the DKP `create cluster` command to create the VM nodes in your cluster directly on a vCenter server. From that point, you can use DKP to provision and manage your cluster.

4.13.6.3.2 Next Step:

vSphere FIPS: Create an Image (see page 494)

4.13.6.4 vSphere FIPS: Create an Image

Creating a base OS image from DVD ISO files is a one-time process. The base OS image file is created in the vSphere Client for use in the vSphere VM template. Therefore, the base OS image is used by Konvoy Image Builder (see page 1626)(KIB) to create a VM template to configure Kubernetes nodes by the DKP vSphere provider.

4.13.6.4.1 Create the Base OS Image

For vSphere, a username is populated by `SSH_USERNAME` and the user can use authorization via `SSH_PASSWORD` or `SSH_PRIVATE_KEY_FILE` environment variables and required by default by packer. This user should have administrator privileges. It is possible to configure a custom user and password when building the OS image, however, that requires the Konvoy Image Builder (KIB) configuration to be overridden (see page 1670).

While creating the base OS image, it is important to take into consideration the following elements:

- Storage configuration: D2iQ recommends customizing disk partitions and not configuring a SWAP partition.
- Network configuration: as KIB must download and install packages, activating the network is required.
- Connect to Red Hat: if using RHEL, registering with Red Hat is required to configure software repositories and install software packages.
- Software selection: D2iQ recommends choosing **Minimal Install**.
- DKP recommends to install with the packages provided by the operating system package managers. Use the version that corresponds to the major version of your operating system.

4.13.6.4.2 Next Step:

The next step is to vSphere FIPS: Create a CAPI VM Template Image (see page 494). The image contains the CAPI and Kubernetes objects.

4.13.6.5 vSphere FIPS: Create a CAPI VM Template

4.13.6.5.1 Prerequisites

- Users need to create a vSphere FIPS: Create an Image (see page 494) before starting this procedure.
- Build image with Konvoy Image Builder (KIB)
- In step 4 of the following section, ensure you use `--overrides overrides/fips.yaml`

4.13.6.5.2 Create a vSphere Template for Your Cluster from a Base OS Image

Using the base OS image created in a previous procedure, DKP creates the new vSphere template directly on the vCenter server.

1. Set the following vSphere environment variables on the bastion VM host:

```
export VSPHERE_SERVER=your_vCenter_APIserver_URL
export VSPHERE_USERNAME=your_vCenter_user_name
export VSPHERE_PASSWORD=your_vCenter_password
```

2. Copy the base OS image file created in the vSphere Client to your desired location on the bastion VM host and make a note of the path and file name.

3. Create an `image.yaml` file and add the following variables for vSphere. DKP uses this file and these variables as inputs in the next step. To customize your `image.yaml` file, refer to this section: Customize your Image (see page 1661).
⚠ **NOTE**: This example is Ubuntu 20.04. You will need to replace OS name below based on your OS. See other default YAML examples[389] for copy and paste below last step.

```
---
download_images: true
build_name: "ubuntu-2004"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-
init-vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: "<VSPHERE_CLUSTER_NAME>"
  datacenter: "<VSPHERE_DATACENTER_NAME>"
  datastore: "<VSPHERE_DATASTORE_NAME>"
  folder: "<VSPHERE_FOLDER>"
  insecure_connection: "false"
  network: "<VSPHERE_NETWORK>"
  resource_pool: "<VSPHERE_RESOURCE_POOL>"
  template: "os-qualification-templates/d2iq-base-Ubuntu-20.04" # change
default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "ubuntu2004-64"
  # goss params
  distribution: "ubuntu"
  distribution_version: "20.04"
# Use following overrides to select the authentication method that can be used
with base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
```

389 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

```
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key
will be ignored
```

4. Create a vSphere VM template with your variation of the following command:

   ```
   konvoy-image build images/ova/<image.yaml>
   ```

   Any additional configurations can be added to this command using `--overrides` flags as shown below:

   a. Any credential overrides: `--overrides overrides.yaml`
   b. for FIPS, add this flag: `--overrides overrides/fips.yaml`
   c. for air-gapped, add this flag: `--overrides overrides/offline-fips.yaml`

5. The Konvoy Image Builder (see page 1652) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. This template contains the required artifacts needed to create a Kubernetes cluster.
   When KIB provisions the OS image[390] successfully, it creates a manifest file. The `artifact_id` field of this file contains the name of the AMI ID (AWS), template name (vSphere), or image name (GCP/Azure), for example:

   ```
   {
         "name": "vsphere-clone",
         "builder_type": "vsphere-clone",
         "build_time": 1644985039,
         "files": null,
         "artifact_id": "konvoy-ova-vsphere-rhel-84-1.21.6-1644983717",
         "packer_run_uuid": "260e8110-77f8-ca94-e29e-ac7a2ae779c8",
         "custom_data": {
           "build_date": "2022-02-16T03:55:17Z",
           "build_name": "vsphere-rhel-84",
           "build_timestamp": "1644983717",
           [...]
         }
       }
   ```

   Recommendation: Now we can now see the template created in our vCenter, it is best to rename it to `dkp-<DKP_VERSION>-k8s-<K8S_VERSION>-<DISTRO>`, like `dkp-2.4.0-k8s-1.24.6-ubuntu` to keep templates organized.

6. Next steps are to deploy a DKP cluster using your vSphere template.

390 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

4.13.6.5.2.1 Additional OS YAML files for copy/paste:

RHEL 8.6

```
---
download_images: true
build_name: "rhel-86"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rhel-8.6" # change default value with your base template name
  vsphere_guest_os_type: "rhel8_64Guest"
  guest_os_type: "rhel8-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "8.6"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Ubuntu 20.04

```
---
download_images: true
build_name: "ubuntu-2004"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
```

```
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-ubuntu-20.04" # change default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "ubuntu2004-64"
  # goss params
  distribution: "ubuntu"
  distribution_version: "20.04"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Rocky Linux 9.1

```
---
download_images: true
build_name: "rocky-91"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rocky-9.1" # change default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "rocky9-64"
  # goss params
  distribution: "rocky"
  distribution_version: "9.1"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
```

```
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

4.13.6.5.2.2 Next Step:

vSphere FIPS: Create the Management Cluster (see page 499)

4.13.6.6 vSphere FIPS: Create the Management Cluster

This page instructions will create a self-managed cluster for use as the Management cluster. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing. Additional considerations exist when using FIPS. Alter the Create a Kubernetes cluster command by using the FIPS example, or similar, when executing the command to create a cluster.

4.13.6.6.1 Deploying a Cluster in FIPS mode

In order to create a cluster in FIPS mode, we must inform the bootstrap controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes.

4.13.6.6.1.1 Supported FIPS Builds

Component	Repository	Version
Kubernetes	docker.io/mesosphere[391]	v1.28.7+fips.0
etcd	docker.io/mesosphere[392]	3.5.10+fips.0

4.13.6.6.2 Name your Cluster

1. Give your cluster a unique name suitable for your environment. The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[393] for more naming information.

391 https://hub.docker.com/layers/mesosphere/kube-apiserver/v1.28.7_d2iq.0/images/sha256-1cafe40f5a17c86aa75574b25dd637bbeb377e2ac703532d72f1118c1e966e28?context=explore
392 https://hub.docker.com/layers/mesosphere/etcd/v3.5.10_fips.0/images/sha256-60da8d0d3b37e71a4fa918ffa7ffd9963d9892b3ba43b8f63119d806f2e585ed?context=explore
393 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

2. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=<my-vsphere-cluster>
```

4.13.6.6.3 Create a New vSphere Kubernetes Cluster

DKP uses the vsphere CSI driver as the default storage provider (see page 110). Use a Kubernetes CSI[394] compatible storage that is suitable for production.

The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.
If you do not set a workspace, it will be created in the `default` workspace, and you need to take additional steps to attach to a workspace later. For instructions on how to do this, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster). (see page 784).

Follow these steps to create the cluster:

1. Use the following command to set the environment variables for vSphere:

```
export VSPHERE_SERVER=example.vsphere.url
export VSPHERE_USERNAME=user@example.vsphere.url
export VSPHERE_PASSWORD=example_password
```

2. Generate the Kubernetes cluster objects by copying and editing this command to include the correct values, including the VM template name you assigned in the previous procedure:

- To increase Dockerhub's rate limit[395] use your Dockerhub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

394 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
395 https://docs.docker.com/docker-hub/download-rate-limit/

- The following example shows a common configuration. See dkp create cluster (see page 1853) reference for the full list of cluster creation options:

```
dkp create cluster vsphere \
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <xxx.yyy.zzz.000> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file <SSH_PUBLIC_KEY_FILE> \
  --resource-pool <RESOURCE_POOL_NAME> \
  --vm-template <TEMPLATE_NAME> \
  --virtual-ip-interface <ip_interface_name> \
  --kubernetes-version=v1.28.7+fips.0 \
  --kubernetes-image-repository=docker.io/mesosphere \
  --etcd-image-repository=docker.io/mesosphere \
  --etcd-version=3.5.10+fips.0 \
  --self-managed
```

Flatcar OS

Flatcar OS (see page 1183) use `--os-hint` to instruct the bootstrap cluster to make some changes related to the installation paths:

```
--os-hint flatcar
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

For bootstrap and custom YAML cluster creation, refer to the Custom Installation and Additional Infrastructure Tools (see page 1050) section of the documentation for vSphere: vSphere Infrastructure (see page 1354) or FIPS 140-2 Compliance (see page 1598) for more FIPS details.

4.13.6.6.3.1 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.13.6.6.3.2 Next Step:

vSphere FIPS: Configure Metal LB (see page 502)

4.13.6.7 vSphere FIPS: Configure Metal LB

If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can proceed to vSphere FIPS: Install Kommander (see page 504) and continue the installation process.

To use MetalLB, create a MetalLB configMap for your vSphere infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

4.13.6.7.1 Layer 2 Configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

- MetalLB IP address ranges/CIDRs should be within the node's primary network subnet (see page 1065).
- MetalLB IP address ranges/CIDRs and node subnet should not conflict with the Kubernetes cluster pod and service subnets.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
```

```
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

When complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.13.6.7.2 BGP configuration

BGP, like any routing protocol, requires a mesh, so you configure BGP between the nodes. If you are in a Data Center, peer with the BGP routers. The instructions for use with Cloud providers is the same. For a standard configuration featuring one BGP router and one IP address range, you need four pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

Extract the kubeconfig and deploy a config map for MetalLB using the following command:

```
dkp get kubeconfig -c ${DKP_CLUSTER_NAME} > ${DKP_CLUSTER_NAME}.conf
```

Deploy MetalLB Configuration with the command below:

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
```

```
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

When complete, run the following `kubectl` command:

```
kubectl apply -f metallb-conf.yaml
```

4.13.6.7.3 Next Step:

vSphere FIPS: Install Kommander (see page 504)

4.13.6.8 vSphere FIPS: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

4.13.6.8.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

4.13.6.8.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

```
dkp install kommander --init > kommander.yaml
```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1541)), etc.

4.13.6.8.1.2 Enable DKP Catalog Applications and Install Kommander

> If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.7.0
```

   ⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.13.6.8.1.3 Next Step:

vSphere FIPS: Verify Install and Log in to the UI (see page 506)

4.13.6.9 vSphere FIPS: Verify Install and Log in to the UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.13.6.9.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

4.13.6.9.2 Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

4.13.6.9.3 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

4.13.6.9.4 Next Step:

vSphere FIPS: Create a Managed Cluster Using the DKP CLI (see page 508)

4.13.6.10 vSphere FIPS: Create a Managed Cluster Using the DKP CLI

Enterprise Gov Advanced

Creating a vSphere FIPS Managed cluster with the DKP CLI assumes that you already fulfilled all of the prerequisites and successfully created a vSphere Management cluster. Use this procedure to create a Managed vSphere cluster.

> ⚠ When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

4.13.6.10.1 Make New Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.13.6.10.2 Name your Cluster

1. Give your cluster a unique name suitable for your environment. The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[396] for more naming information.
2. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=<my-managed-vsphere-cluster>
```

396 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

4.13.6.10.3 Create a New vSphere Kubernetes Cluster

⚠ DKP uses the vsphere CSI driver as the default storage provider (see page 110). Use a Kubernetes CSI[397] compatible storage that is suitable for production.

The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.
If you do not set a workspace, it will be created in the `default` workspace, and you need to take additional steps to attach to a workspace later. For instructions on how to do this, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster). (see page 784).

Follow these steps to create the cluster:

1. Use the following command to set the environment variables for vSphere:

```
export VSPHERE_SERVER=example.vsphere.url
export VSPHERE_USERNAME=user@example.vsphere.url
export VSPHERE_PASSWORD=example_password
```

2. Generate the Kubernetes cluster objects by copying and editing this command to include the correct values, including the VM template name you assigned in the previous procedure:

- To increase Dockerhub's rate limit[398] use your Dockerhub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.
- The following example shows a common configuration. See dkp create cluster (see page 1853) reference for the full list of cluster creation options:

```
dkp create cluster vsphere \
```

397 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
398 https://docs.docker.com/docker-hub/download-rate-limit/

```
--cluster-name=${MANAGED_CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--kubeconfig=<management-cluster-kubeconfig-path> \
--namespace ${WORKSPACE_NAMESPACE}
--network <NETWORK_NAME> \
--control-plane-endpoint-host <xxx.yyy.zzz.000> \
--data-center <DATACENTER_NAME> \
--data-store <DATASTORE_NAME> \
--folder <FOLDER_NAME> \
--server <VCENTER_API_SERVER_URL> \
--ssh-public-key-file <SSH_PUBLIC_KEY_FILE> \
--resource-pool <RESOURCE_POOL_NAME> \
--vm-template <TEMPLATE_NAME> \
--virtual-ip-interface <ip_interface_name> \
--kubernetes-version=v1.28.7+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
--etcd-image-repository=docker.io/mesosphere \
--etcd-version=3.5.10+fips.0 \
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

4.13.6.10.4 Retrieve the kubeconfig and Explore New vSphere Cluster

Follow these steps:

1. Fetch the kubeconfig file with the command:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} --kubeconfig
   <management-cluster-kubeconfig-path> -n ${WORKSPACE_NAMESPACE} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

2. List the nodes with the following command:

   ```
   kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get nodes
   ```

3. List the pods with the following command:

NOTE: Wait for the Status to move to Ready while `calico-node` pods are being deployed.

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get pods -A
```

4.13.6.10.4.1 Manually Attach a DKP CLI Cluster to the Management Cluster

> ⚠ These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

   ```
   echo ${MANAGED_CLUSTER_NAME}
   ```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
   NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

   ```
   kubectl get workspaces -A
   ```

5. Set the WORKSPACE_NAMESPACE environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace-namespace>
   ```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

```
kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
template='{{.data.value}}{{ "\n"}}'
```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

If you have existing clusters or want to create other new clusters to attach, there are many ways to attach a cluster with various requirements and restrictions. To see all the options, visit the section in documentation Day 2 - Attach an Existing Kubernetes Cluster (see page 784).

At this point, you can create more clusters, perform other configuration tasks (see page 1050), or proceed to Day 2 Operations (see page 590).

4.13.7 vSphere FIPS Air-gapped Install

This section provides instructions to install DKP in a vSphere with FIPS in an **air-gapped** environment.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

For air-gapped, ensure you have downloaded (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , so you can extract the tarball on the page with those instructions.

4.13.7.1 Further vSphere Prerequisites

- vSphere: VMware Prerequisites (see page 446)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430985/vSphere+Prerequisites+-+Storage+Options (see page 445)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430977/vSphere+Prerequisites+-+Minimum+User+Permissions (see page 442)

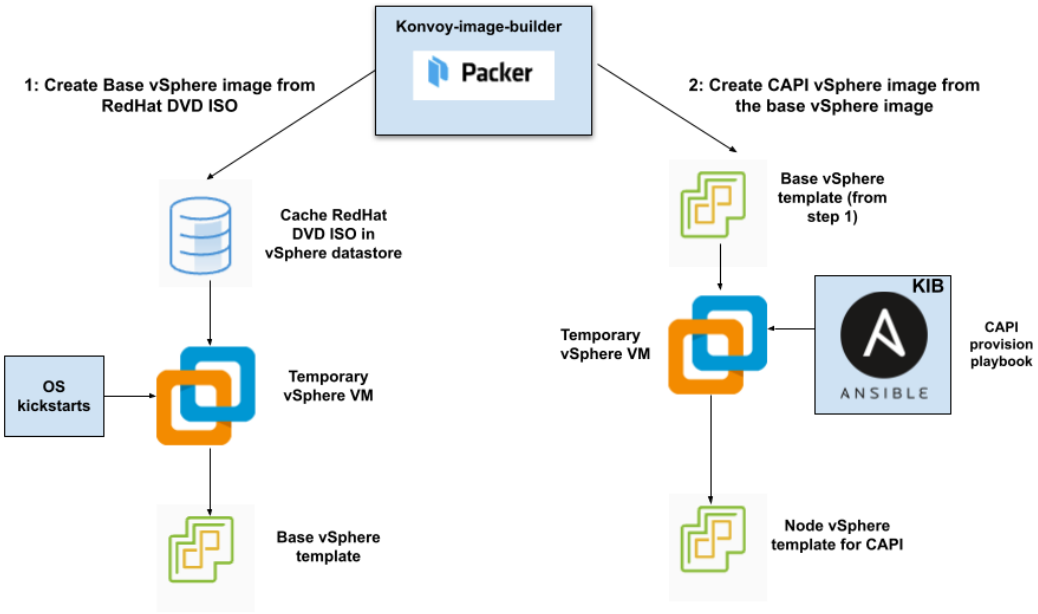
4.13.7.2 Next Step:

vSphere FIPS Air-gapped: Image Creation Overview (see page 514)

4.13.7.3 vSphere FIPS Air-gapped: Image Creation Overview

4.13.7.3.1 Image Creation Process

This diagram illustrates the image creation process:



The workflow on the left shows the creation of a base OS image in the vCenter vSphere client using inputs from Packer. The workflow on the right shows how DKP uses that same base OS image to create CAPI-enabled VM images for your cluster.

After creating the base image, the DKP image builder uses it to create a CAPI-enabled vSphere template that includes the Kubernetes objects for the cluster. You can use that resulting template with the DKP `create cluster` command to create the VM nodes in your cluster directly on a vCenter server. From that point, you can use DKP to provision and manage your cluster.

4.13.7.3.2 Next Step:

vSphere FIPS Air-gapped: Create an Image (see page 515)

4.13.7.4 vSphere FIPS Air-gapped: Create an Image

Creating a base OS image from DVD ISO files is a one-time process. The base OS image file is created in the vSphere Client for use in the vSphere VM template. Therefore, the base OS image is used by Konvoy Image Builder (see page 1626)(KIB) to create a VM template to configure Kubernetes nodes by the DKP vSphere provider.

4.13.7.4.1 Create the Base OS Image

For vSphere, a username is populated by `SSH_USERNAME` and the user can use authorization via `SSH_PASSWORD` or `SSH_PRIVATE_KEY_FILE` environment variables and required by default by packer. This user should have administrator privileges. It is possible to configure a custom user and password when building the OS image, however, that requires the Konvoy Image Builder (KIB) configuration to be overridden (see page 1670).

While creating the base OS image, it is important to take into consideration the following elements:

- Storage configuration: D2iQ recommends customizing disk partitions and not configuring a SWAP partition.
- Network configuration: as KIB must download and install packages, activating the network is required.
- Connect to Red Hat: if using RHEL, registering with Red Hat is required to configure software repositories and install software packages.
- Software selection: D2iQ recommends choosing **Minimal Install**.
- DKP recommends to install with the packages provided by the operating system package managers. Use the version that corresponds to the major version of your operating system.

4.13.7.4.2 Next Step:

vSphere FIPS Air-gapped: Docker Registry (see page 516)

4.13.7.5 vSphere FIPS Air-gapped: Load the Registry

Before creating an air-gapped Kubernetes cluster, you need to load the required images in a local registry for the Konvoy component. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1606) page for more information.

1. If not already done in prerequisites, download (see page 76) the air-gapped bundle `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap cluster, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

```
cd dkp-v2.8.1
```

3. Set an environment variable with your registry address and any other needed variables using this command:

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
export REGISTRY_CA=<path to the cacert file on the bastion>
```

4. Execute the following command to load the air-gapped image bundle into your private registry using any of the relevant flags to apply variables above:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --
to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-
registry-password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

4.13.7.5.1 Kommander Load Images

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

4.13.7.5.2 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

4.13.7.5.3 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME}
--to-registry-password=${REGISTRY_PASSWORD}
```

4.13.7.5.4 Next Step:

vSphere FIPS Air-gapped: Create a CAPI VM Template Image (see page 518)

4.13.7.6 vSphere FIPS Air-gapped: Create a CAPI VM Template

You must have at least one image before creating a new cluster. As long as you have an image, this step in your configuration is not required each time since that image can be used to spin up a new cluster. However, if you need different images for different environments or providers, you will need to create a new custom image.

Using KIB (see page 1626), you can create your VM template without requiring access to the internet by providing an additional `--override` flag.

1. Assuming you have downloaded (see page 76) `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz`, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz && cd dkp-v2.8.0/kib
   ```

2. Follow the instructions to build a vSphere template below and set the override `--overrides overrides/offline.yaml` flag.

4.13.7.6.1 Create a vSphere Template for Your Cluster from a Base OS Image

Using the base OS image created in a previous procedure, DKP creates the new vSphere template directly on the vCenter server.

1. Set the following vSphere environment variables on the bastion VM host:

   ```
   export VSPHERE_SERVER=your_vCenter_APIserver_URL
   ```

```
export VSPHERE_USERNAME=your_vCenter_user_name
export VSPHERE_PASSWORD=your_vCenter_password
```

2. Copy the base OS image file created in the vSphere Client to your desired location on the bastion VM host and make a note of the path and file name.
3. Create an `image.yaml` file and add the following variables for vSphere. DKP uses this file and these variables as inputs in the next step. To customize your `image.yaml` file, refer to this section: Customize your Image (see page 1661).
   ⚠ **NOTE**: This example is Ubuntu 20.04. You will need to replace OS name below based on your OS. See other default YAML examples[399] for copy and paste below last step.

```
---
download_images: true
build_name: "ubuntu-2004"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-
init-vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: "<VSPHERE_CLUSTER_NAME>"
  datacenter: "<VSPHERE_DATACENTER_NAME>"
  datastore: "<VSPHERE_DATASTORE_NAME>"
  folder: "<VSPHERE_FOLDER>"
  insecure_connection: "false"
  network: "<VSPHERE_NETWORK>"
  resource_pool: "<VSPHERE_RESOURCE_POOL>"
  template: "os-qualification-templates/d2iq-base-Ubuntu-20.04" # change
default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "ubuntu2004-64"
  # goss params
  distribution: "ubuntu"
  distribution_version: "20.04"
# Use following overrides to select the authentication method that can be used
with base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key
will be ignored
```

399 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

4. Create a vSphere VM template with your variation of the following command:

```
konvoy-image build images/ova/<image.yaml>
```

Any additional configurations can be added to this command using `--overrides` flags as shown below:

   a. Any credential overrides: `--overrides overrides.yaml`
   b. for FIPS, add this flag: `--overrides overrides/fips.yaml`
   c. for air-gapped, add this flag: `--overrides overrides/offline-fips.yaml`

5. The Konvoy Image Builder (see page 1652) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. This template contains the required artifacts needed to create a Kubernetes cluster.
When KIB provisions the OS image[400] successfully, it creates a manifest file. The `artifact_id` field of this file contains the name of the AMI ID (AWS), template name (vSphere), or image name (GCP/Azure), for example:


```
{
      "name": "vsphere-clone",
      "builder_type": "vsphere-clone",
      "build_time": 1644985039,
      "files": null,
      "artifact_id": "konvoy-ova-vsphere-rhel-84-1.21.6-1644983717",
      "packer_run_uuid": "260e8110-77f8-ca94-e29e-ac7a2ae779c8",
      "custom_data": {
        "build_date": "2022-02-16T03:55:17Z",
        "build_name": "vsphere-rhel-84",
        "build_timestamp": "1644983717",
        [...]
      }
    }
```


Recommendation: Now we can now see the template created in our vCenter, it is best to rename it to `dkp-<DKP_VERSION>-k8s-<K8S_VERSION>-<DISTRO>`, like `dkp-2.4.0-k8s-1.24.6-ubuntu` to keep templates organized.

6. Next steps are to deploy a DKP cluster using your vSphere template.

4.13.7.6.1.1 Additional OS YAML file examples:

RHEL 8.6

400 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

```
---
download_images: true
build_name: "rhel-86"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rhel-8.6" # change default value with your base template name
  vsphere_guest_os_type: "rhel8_64Guest"
  guest_os_type: "rhel8-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "8.6"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Ubuntu 20.04

```
---
download_images: true
build_name: "ubuntu-2004"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
```

```
  resource_pool: ""
  template: "base-ubuntu-20.04" # change default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "ubuntu2004-64"
  # goss params
  distribution: "ubuntu"
  distribution_version: "20.04"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Rocky Linux 9.1

```
---
download_images: true
build_name: "rocky-91"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "base-rocky-9.1" # change default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "rocky9-64"
  # goss params
  distribution: "rocky"
  distribution_version: "9.1"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

4.13.7.6.1.2 Next Step

vSphere FIPS Air-gapped: Create the Management Cluster (see page 523)

4.13.7.7 vSphere FIPS Air-gapped: Create the Management Cluster

The instructions on this page will generate a self-managed air-gapped cluster to be used as the Management cluster. A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing.

4.13.7.7.1 Deploying a Cluster in FIPS mode

In order to create a cluster in FIPS mode, we must inform the bootstrap controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes.

4.13.7.7.1.1 Supported FIPS Builds

Component	Repository	Version
Kubernetes	docker.io/mesosphere[401]	v1.28.7+fips.0
etcd	docker.io/mesosphere[402]	3.5.10+fips.0

4.13.7.7.2 Name your Cluster

1. Give your cluster a unique name suitable for your environment. The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[403] for more naming information.
2. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=<my-vsphere-cluster>
```

401 https://hub.docker.com/layers/mesosphere/kube-apiserver/v1.28.7_d2iq.0/images/sha256-1cafe40f5a17c86aa75574b25dd637bbeb377e2ac703532d72f1118c1e966e28?context=explore
402 https://hub.docker.com/layers/mesosphere/etcd/v3.5.10_fips.0/images/sha256-60da8d0d3b37e71a4fa918ffa7ffd9963d9892b3ba43b8f63119d806f2e585ed?context=explore
403 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

DKP uses the vsphere CSI driver as the default storage provider (see page 110). Use a Kubernetes CSI[404] compatible storage that is suitable for production.

4.13.7.7.2.1 Create a New vSphere Kubernetes Cluster

The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.
If you do not set a workspace, it will be created in the `default` workspace, and you need to take additional steps to attach to a workspace later. For instructions on how to do this, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster). (see page 784).

Use the following steps to create a new, air-gapped vSphere cluster.

1. Configure your cluster to use an existing local registry as a mirror when attempting to pull images: **IMPORTANT:** The image must be created by Konvoy Image Builder (see page 1652) in order to use the registry mirror feature.

   ```
   export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
   export REGISTRY_CA=<path to the CA on the bastion>
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   ```

   - `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
   - `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
   - `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
   - `REGISTRY_PASSWORD` : optional if username is not set.

2. Load the image, using either the `docker` or `podman` command:

   ```
   docker load -i konvoy-bootstrap-image-v2.8.0.tar
   ```

   OR

404 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

```
podman load -i konvoy-bootstrap-image-v2.8.0.tar
```

3. Create a Kubernetes cluster by copying the following command and substituting the valid values for your environment:

```
dkp create cluster vsphere
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <CONTROL_PLANE_IP> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file </path/to/key.pub> \
  --resource-pool <RESOURCE_POOL_NAME> \
  --vm-template konvoy-ova-vsphere-os-release-k8s_release-vsphere-timestamp \
  --virtual-ip-interface <ip_interface_name> \
  --extra-sans "127.0.0.1" \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --kubernetes-version=v1.28.7+fips.0 \
  --kubernetes-image-repository=docker.io/mesosphere \
  --etcd-image-repository=docker.io/mesosphere --etcd-version=3.5.10+fips.0 \
  --self-managed
```

Flatcar OS

Flatcar OS (see page 1183) use `--os-hint` to instruct the bootstrap cluster to make some changes related to the installation paths:

```
--os-hint flatcar
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

For bootstrap and custom YAML cluster creation, refer to the Custom Installation and Additional Infrastructure Tools (see page 1050) section of the documentation for vSphere: Install vSphere in an Air-Gapped Environment (see page 1404) or FIPS 140-2 Compliance (see page 1598) for more FIPS details.

4.13.7.7.2.2 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

As they progress, the controllers create Events, which you can also monitor using the command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, this example uses `grep`. You can also use separate commands to get Events for specific objects, such as `kubectl get events --field-selector involvedObject.kind="VSphereCluster"` and `kubectl get events --field-selector involvedObject.kind="VSphereMachine"`.

4.13.7.7.2.3 Next Step:

vSphere FIPS Air-gapped: Configure Metal LB (see page 526)

4.13.7.8 vSphere FIPS Air-gapped: Configure Metal LB

If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can proceed to vSphere FIPS Air-gapped: Install Kommander (see page 529) and continue the installation process.

To use MetalLB, create a MetalLB configMap for your vSphere infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

4.13.7.8.1 Layer 2 Configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

- MetalLB IP address ranges/CIDRs should be within the node's primary network subnet (see page 1065).
- MetalLB IP address ranges/CIDRs and node subnet should not conflict with the Kubernetes cluster pod and service subnets.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

When complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

4.13.7.8.2 BGP configuration

BGP, like any routing protocol, requires a mesh, so you configure BGP between the nodes. If you are in a Data Center, peer with the BGP routers. The instructions for use with Cloud providers is the same. For a standard configuration featuring one BGP router and one IP address range, you need four pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,

- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

Extract the kubeconfig and deploy a config map for MetalLB using the following command:

```
dkp get kubeconfig -c ${DKP_CLUSTER_NAME} > ${DKP_CLUSTER_NAME}.conf
```

Deploy MetalLB Configuration with the command below:

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

When complete, run the following `kubectl` command:

```
kubectl apply -f metallb-conf.yaml
```

4.13.7.8.3 Next Step:

vSphere FIPS Air-gapped: Install Kommander (see page 529)

4.13.7.9 vSphere FIPS Air-gapped: Install Kommander

4.13.7.9.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry: Air-gapped Environments (see page 1536).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

4.13.7.9.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init --airgapped > kommander.yaml
   ```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1541)), etc.

4.13.7.9.1.2 Enable DKP Catalog Applications and Install Kommander in an Air-gapped Environment

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add the following values to enable DKP Catalog Applications:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./dkp-catalog-applications-v2.8.1.tar.gz
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.1.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.1.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.8.1.tar.gz
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.13.7.9.1.3 Next Step:

vSphere FIPS Air-gapped: Verify Install and Log in to the UI (see page 531)

4.13.7.10 vSphere FIPS Air-gapped: Verify Install and Log in to the UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

> **NOTE**: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.13.7.10.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a "broken" release state, such as "exhausted" or "another rollback/release in progress", trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

4.13.7.10.2 Log in to the UI with Kommander

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

4.13.7.10.2.1 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

4.13.7.10.2.2 Next Step:

vSphere FIPS Air-gapped: Create a Managed Cluster Using the DKP CLI (see page 533)

4.13.7.11 vSphere FIPS Air-gapped: Create a Managed Cluster Using the DKP CLI

Enterprise Gov Advanced

Creating an air-gapped vSphere FIPS Managed cluster with the DKP CLI assumes that you already fulfilled all of the prerequisites and successfully created a vSphere Management cluster. Use this procedure to create a Managed vSphere cluster.

⚠ When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

4.13.7.11.1 Make New Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

```
kubectl get workspace -A
```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

4.13.7.11.2 Name your Cluster

1. Give your cluster a unique name suitable for your environment.
2. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=<my-managed-vsphere-cluster>
```

DKP uses local static provisioner as the default storage provider (see page 110). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[405] compatible storage that is suitable for production.

- You can choose from any of the storage options[406] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[407].

4.13.7.11.3 Create a New vSphere Kubernetes Cluster

The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.
If you do not set a workspace, it will be created in the `default` workspace, and you need to take additional steps to attach to a workspace later. For instructions on how to do this, see Attach a Kubernetes Cluster (see page 784).

Use the following steps to create a new, air-gapped vSphere cluster.

405 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
406 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
407 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

1. Configure your cluster to use an existing registry as a mirror when attempting to pull images:
   IMPORTANT: The image must be created by Konvoy Image Builder (see page 1626) so that it uses the registry mirror feature.

   ```
   export REGISTRY_URL=<https/http>://<registry-address>:<registry-port>
   export REGISTRY_CA=<path to the CA on the bastion>
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   ```

   - `REGISTRY_URL` : the address of an existing Docker registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
   - `REGISTRY_CA` : (optional) the path on the bastion machine to the Docker registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
   - `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
   - `REGISTRY_PASSWORD` : optional if username is not set.

2. Create a Kubernetes cluster by copying the following command and substituting the valid values for your environment:

```
dkp create cluster vsphere \
  --cluster-name ${MANAGED_CLUSTER_NAME} \
  --additional-tags=owner=$(whoami) \
  --namespace ${WORKSPACE_NAMESPACE}
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <CONTROL_PLANE_IP> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file </path/to/key.pub> \e
  --resource-pool <RESOURCE_POOL_NAME> \
  --vm-template konvoy-ova-vsphere-os-release-k8s_release-vsphere-timestamp \
  --virtual-ip-interface <ip_interface_name> \
  --extra-sans "127.0.0.1" \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --kubernetes-version=v1.28.7+fips.0 \
  --kubernetes-image-repository=docker.io/mesosphere \
  --etcd-image-repository=docker.io/mesosphere \
  --etcd-version=3.5.10+fips.0 \
  --kubeconfig=<management-cluster-kubeconfig-path> \
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

4.13.7.11.4 Retrieve the kubeconfig and Explore New vSphere Cluster

Follow these steps:

1. Fetch the kubeconfig file with the command:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} --kubeconfig
<management-cluster-kubeconfig-path> -n ${WORKSPACE_NAMESPACE} > $
{MANAGED_CLUSTER_NAME}.conf
```

2. List the nodes with the following command:

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get nodes
```

3. List the pods with the following command:

NOTE: Wait for the Status to move to Ready while `calico-node` pods are being deployed.

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get pods -A
```

4.13.7.11.4.1 Manually Attach a DKP CLI Cluster to the Management Cluster

These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

   ```
   echo ${MANAGED_CLUSTER_NAME}
   ```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
   NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

   ```
   kubectl get workspaces -A
   ```

5. Set the WORKSPACE_NAMESPACE environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace-namespace>
   ```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

   ```
   kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
   template='{{.data.value}}{{ "\n"}}'
   ```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: <your-managed-cluster-name>-kubeconfig
     labels:
       cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
   type: cluster.x-k8s.io/secret
   data:
     value: <value-you-copied-from-secret-above>
   ```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

If you have existing clusters or want to create other new clusters to attach, there are many ways to attach a cluster with various requirements and restrictions. To see all the options, visit the section in documentation Day 2 - Attach an Existing Kubernetes Cluster (see page 784).

At this point, you can create more clusters, perform other configuration tasks (see page 1050), or proceed to Day 2 Operations (see page 590).

4.14 VMware Cloud Director Install Options

For VMware Cloud Director (VCD), an installation scenario is provided for you in this location. Remember, there are always more options in the Additional Infrastructure Configurations (see page 1354) sections, but this will get you operative in the most common scenarios.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

4.14.1 Specific to VMware Cloud Director:

4.14.1.1 CPU and Memory

1. For CPU and Memory, the VCD **Provider** must create the appropriate VM Sizing Policies.
2. The **Provider** must reference these VM Sizing Policies when they create the cluster, using the `--control-plane-sizing-policy` and `--worker-sizing-policy` flags.
3. See https://docs.vmware.com/en/VMware-Cloud-Director/10.4/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-73C48A9C-79AF-402C-9746-4C81CAC2B35A.html regarding parameters like vCPU Speed or CPU Reservation Guarantee that require consideration in VM Sizing Policy.

4.14.1.2 Storage

For storage, the VCD **Provider** must follow Create a Base OS image in vSphere vCenter (see page 1366) when they create the KIB base image. In VCD, the base image determines the minimum storage available for the VM.

For more information about VCD parameters that can affect performance, refer to VMware Documentation: https://docs.vmware.com/en/VMware-Cloud-Director/10.4/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-73C48A9C-79AF-402C-9746-4C81CAC2B35A.html

Below are all the supported environment variable combinations: (Refer to this page to check if your OS is a Supported Operating System (see page 67).)

- Cloud Director Service Provider (see page 539)

4.14.1.3 Next Step:

Cloud Director Service Provider (see page 539)

4.14.2 Cloud Director Service Provider

There is not a "**Day 1 - Basic Install**" for VMWare Cloud Director since each tenant will have different needs. For Managed Service Providers (MSPs), please refer to Cloud Director for Service Providers (see page 1445) section of the documentation regarding installation plus image and OVA export and import for your Tenant Organizations.

Before continuing to install DKP on Cloud Director, verify that your VMware vSphere Client environment is running vSphere Client version[408]v6.7.x with Update 3 or later version with ESXi[409]. You must be able to reach the vSphere API endpoint from where the Konvoy command line interface (CLI) runs and have a vSphere

408 https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.vsphere.vm_admin.doc/GUID-55238059-912E-411F-A0E9-A7A536972A91.html
409 https://docs.vmware.com/en/VMware-vSphere/index.html

account containing Administrator privileges. A RedHat® subscription is required with username and password for downloading DVD ISOs and valid vSphere values for the following: vCenter API server URL Datacenter name Zone name that contains ESXi hosts for your cluster's nodes.

4.14.2.1 If Required - Further vSphere Prerequisites

- vSphere: VMware Prerequisites (see page 446)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430985/vSphere+Prerequisites+-+Storage+Options (see page 445)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430977/vSphere+Prerequisites+-+Minimum+User+Permissions (see page 442)

4.14.2.2 Overview

VMware Cloud Director (VCD) provides a private cloud like experience in data centers. VCD allows creation of virtual data centers for a more self-service of administration tasks. An overview of steps has been provided to help.

The overall process for configuring VCD and DKP together includes the following steps:

1. Configure vSphere to provide the needed elements, described in the vSphere Prerequisites: All Installation Types (see page 442).
2. **For air-gapped environments:** Create a bastion VM host (see page 1068).
3. Create a base OS image (for use in the OVA package containing the disk images packaged with the OVF).
4. Create a CAPI VM image template that uses the base OS image and adds the needed Kubernetes cluster components.
5. Create a Tenant Organization
6. Configure Virtual Data Centers (VDCs)
7. MSP will uploaded the appropriate OVA to tenant organization's catalog
    a. MSP uses `konvoy-image build vsphere` to create an OVA
    b. MSP creates a vApp from the OVA
    c. Before the MSP creates a cluster in a VCD tenant, it makes this vApp available to that tenant
8. User must be able to reach the VCD API endpoint from where the DKP CLI runs
9. Create a new self-managing DKP cluster on VCD.
10. Install Kommander
11. Verify and log in to the UI

4.14.2.3 Next Step:

For MSPs, please refer to the VMware Cloud Director Infrastructure (see page 1444) section of the documentation regarding installation and Tenant Organization creation.

4.15 Azure Install Options

For an environment that is Azure, an install is provided for you in this one location.

Remember, there are always more options for custom YAML in the Custom Installation and Additional Infrastructure Tools (see page 1050) section under Azure Infrastructure (see page 1296), but this will get you operative in the most common scenario.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

Below are the supported environment variable combinations: (Refer to this page to check if your OS is a Supported Operating System (see page 67).)

- Azure Install (see page 541)

Additional Resource Information specific to Azure is below.

- **Control plane nodes** - DKP on Azure defaults to deploying a `Standard_D4s_v3` virtual machine with an 128 GiB volume for the OS and an 80GiB volume for etcd storage, which meets the above resource requirements (see page 119).
- **Worker nodes** - DKP on Azure defaults to deploying a `Standard_D8s_v3` virtual machine with an 80 GiB volume for the OS, which meets the above resource requirements (see page 119).

4.15.1 Azure Install

For an environment that is on the Azure Infrastructure, install options are provided for you in this one location. Remember, there are always more options in the Azure Infrastructure (see page 1296) section under Custom Installation and Additional Infrastructure Tools (see page 1050) , but this will get you up and running in the most common scenario.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

4.15.1.1 Azure Prerequisites

1. Sign in to Azure:

```
az login
```


```
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "id": "b1234567-abcd-11a1-a0a0-1234a5678b90",
    "isDefault": true,
    "managedByTenants": [],
    "name": "D2iQ Developer Subscription",
    "state": "Enabled",
    "tenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "user": {
      "name": "user@azuremesosphere.onmicrosoft.com",
      "type": "user"
    }
  }
]
```


2. Create an Azure Service Principal (SP) by running the following commands:
   a. If you have more than one Azure account, run this command to identify your account:

   ```
   echo $(az account show --query id -o tsv)
   ```

   b. Run this command to ensure you are pointing to the correct Azure subscription ID:

   ```
   az account set --subscription "D2iQ Developer Subscription"
   ```

   c. If an SP with the name exists, this command rotates the password.

   ```
   az ad sp create-for-rbac --role contributor --name "$(whoami)-konvoy" --
   scopes=/subscriptions/$(az account show --query id -o tsv) --query
   "{ client_id: appId, client_secret: password, tenant_id: tenant }"
   ```


   ```
   {
     "client_id": "7654321a-1a23-567b-b789-0987b6543a21",
     "client_secret": "Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C",
     "tenant_id": "a1234567-b132-1234-1a11-1234a5678b90"
   }
   ```


3. Set the `AZURE_CLIENT_SECRET` environment variable:

```
export AZURE_CLIENT_SECRET="<azure_client_secret>" #
Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C
export AZURE_CLIENT_ID="<client_id>"               # 7654321a-1a23-567b-
b789-0987b6543a21
export AZURE_TENANT_ID="<tenant_id>"               # a1234567-b132-1234-1a11-12
34a5678b90
export AZURE_SUBSCRIPTION_ID="<subscription_id>"   # b1234567-abcd-11a1-
a0a0-1234a5678b90
```

4. Ensure you have an override file (see page 1670) to configure specific attributes of your Azure image.

4.15.1.2 Next Step:

Azure: Create Image (see page 543)

4.15.1.3 Azure: Create Image

This procedure describes how to use the Konvoy Image Builder (see page 1626) (KIB) to create a Cluster API[410] compliant Azure Virtual Machine (VM) Image. The VM Image contains the base operating system you specify and all the necessary Kubernetes components. The Konvoy Image Builder uses variable `overrides` to specify the base image and container images to use in your new Azure VM image.

The default Azure image is not recommended for use in production. We suggest using KIB for Azure to build the image in order to take advantage of enhanced cluster operations. Explore the Customize your Image (see page 1661) topic for more options.
For more information regarding using the image in creating clusters, refer to the Azure Create a New Cluster (see page 1314) section of the documentation.

4.15.1.3.1 Prerequisites

Before you begin, you must:

- Download the Konvoy Image Builder (see page 1626) bundle for your version of DKP.
- Check the Supported Infrastructure Operating Systems (see page 67)
- Check the Supported Kubernetes Version (see page 1706) for your Provider.
- Create a working `Docker` setup.

410 https://cluster-api.sigs.k8s.io/

4.15.1.3.2 Extract the KIB Bundle

Extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION_$OS` folder. The bundled version of `konvoy-image` contains an embedded `docker` image that contains all the requirements for building.

> The `konvoy-image` binary and all supporting folders are also extracted. When extracted, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

4.15.1.3.3 Build the Image

Run the `konvoy-image` command to build and validate the image.

```
konvoy-image build azure --client-id ${AZURE_CLIENT_ID} --tenant-id $
{AZURE_TENANT_ID} --overrides override-source-image.yaml images/azure/ubuntu-2004.yam
l
```

By default, the image builder builds in the `westus2` location. To specify another location set the `--location` flag (shown in example below is how to change the location to `eastus`):

```
konvoy-image build azure --client-id ${AZURE_CLIENT_ID} --tenant-id $
{AZURE_TENANT_ID} --location eastus --overrides override-source-image.yaml images/
azure/centos-7.yaml
```

When the command is complete, the image id is printed and written to the `./packer.pkr.hcl` file. This file has an `artifact_id` field whose value provides the name of the image. You should then specify this image id when creating the cluster.

4.15.1.3.4 Image Gallery

By default Konvoy Image Builder will create a Resource Group, Gallery, and Image Name to store the resulting image in. To specify a specific Resource Group, Gallery, or Image Name flags may be specified:

```
--gallery-image-locations string     a list of locations to publish the image
(default same as location)
--gallery-image-name string          the gallery image name to publish the image to
```

```
--gallery-image-offer string          the gallery image offer to set (default "dkp")
--gallery-image-publisher string      the gallery image publisher to set (default
"dkp")
--gallery-image-sku string            the gallery image sku to set
--gallery-name string                 the gallery name to publish the image in
(default "dkp")
--resource-group string               the resource group to create the image in
(default "dkp")
```

When creating your cluster, you will then add this flag during the create process for your custom image: `--compute-gallery-id "<Managed Image Shared Image Gallery Id>"`. The SKU and Image Name will default to the values found in the image YAML.

4.15.1.3.4.1 More customization options:

- See Create a New Azure Cluster (see page 543) for specific consumption of image commands.
- See KIB for Azure (see page 1642) for more flags and Rocky Linux Marketplace.

4.15.1.3.5 Next Step:

Azure: Cluster Creation (see page 545)

4.15.1.4 Azure: Create the Management Cluster

4.15.1.4.1 Name your cluster

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<azure-example>
```

4.15.1.4.2 Encode your Azure Credential Variables:

Base64 encode the Azure environment variables set in the Azure install prerequisites (see page 541) step:

```
export AZURE_SUBSCRIPTION_ID_B64="$(echo -n "${AZURE_SUBSCRIPTION_ID}" | base64 | tr
-d '\n')"
export AZURE_TENANT_ID_B64="$(echo -n "${AZURE_TENANT_ID}" | base64 | tr -d '\n')"
export AZURE_CLIENT_ID_B64="$(echo -n "${AZURE_CLIENT_ID}" | base64 | tr -d '\n')"
export AZURE_CLIENT_SECRET_B64="$(echo -n "${AZURE_CLIENT_SECRET}" | base64 | tr -d
'\n')"
```

DKP uses the Azure CSI driver as the default storage provider (see page 110). Use a Kubernetes CSI[411] compatible storage that is suitable for production.

4.15.1.4.3 Create an Azure Kubernetes Cluster

If you use thee instructions to create an Azure cluster using the DKP default settings, your cluster is deployed with 3 control plane nodes and 4 worker nodes.

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools to ensure your cluster has nodes deployed in multiple Availability Zones.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

1. Generate the Kubernetes cluster objects. The following example shows a common configuration. See dkp create cluster azure (see page 1853) reference for the full list of cluster creation options.

   ```
   dkp create cluster azure \
   --cluster-name=${CLUSTER_NAME} \
   --self-managed
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

   Output is similar to below:

   ```
   Generating cluster resources
   ```

   A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing. As part of the underlying processing using the `--self-managed` flag, the DKP CLI:

   - creates a bootstrap cluster
   - creates a workload cluster

411 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

- moves CAPI controllers from the bootstrap cluster to the workload cluster, making it self-managed
- deletes the bootstrap cluster

To understand how this process works step by step, you can find a customizable Create a Custom Azure Cluster (see page 1314) under Custom Installation and Additional Infrastructure Tools (see page 1050).

4.15.1.4.4 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.15.1.4.5 Known Limitations

The Konvoy version used to create a bootstrap cluster must match the Konvoy version used to create a workload cluster.

- Konvoy supports deploying one workload cluster.
- Konvoy generates a set of objects for one Node Pool.
- Konvoy does not validate edits to cluster objects.

4.15.1.4.6 Next Step:

Azure: Install Kommander (see page 547)

4.15.1.5 Azure: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

4.15.1.5.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

4.15.1.5.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

```
export CLUSTER_NAME=<your-management-cluster-name>
```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1558) for the deployment:

```
dkp install kommander --init > kommander.yaml
```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1541)), etc.

4.15.1.5.1.2 Enable DKP Catalog Applications and Install Kommander

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.7.0
```

   ⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.15.1.5.1.3 Next Step:

Azure: Verify Install and Log in to the UI (see page 549)

4.15.1.6 Azure: Verify Install and Log in to the UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.15.1.6.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

4.15.1.6.2 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

Now you can create additional new clusters (see page 551), or proceed to Day 2 Operations in the link.

4.15.1.6.3 Next Step:

Azure: Subsequent New Clusters (see page 551)

4.15.1.7 Azure: Create a Managed Cluster Using the DKP CLI

Enterprise Gov Advanced

In the previous step, the new cluster was created as Self-managed which allows it to be a Management cluster. Subsequent new clusters are not self-managed as they will likely be Managed or Attached clusters to this Management Cluster.

When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

Your new, managed cluster needs to be part of either a new Workspace or an existing Workspace under a management cluster. Create or locate a workspace name to get started with your managed cluster.

4.15.1.7.1 Make New Managed Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.15.1.7.2 Name your Cluster

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

   ```
   export MANAGED_CLUSTER_NAME=<azure-additional>
   ```

4.15.1.7.3 Use DKP CLI

The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.
If you do not set a workspace, it will be created in the `default` workspace, and you need to

take additional steps to attach to a workspace later. For instructions on how to do this, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster). (see page 784).

Execute this command to create an additional cluster without the `self-managed` flag:

```
dkp create cluster azure --cluster-name=${MANAGED_CLUSTER_NAME} \
--namespace=${WORKSPACE_NAMESPACE} \
--additional-tags=owner=$(whoami) \
--kubeconfig=<management-cluster-kubeconfig-path> \
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

4.15.1.7.4 Retrieve the kubeconfig and Explore New Azure Cluster

Follow these steps:

1. Fetch the kubeconfig file with the command:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} --kubeconfig
   <management-cluster-kubeconfig-path> -n ${WORKSPACE_NAMESPACE} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

2. List the nodes with the following command:

   ```
   kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get nodes
   ```

3. List the pods with the following command:

NOTE: Wait for the Status to move to Ready while `calico-node` pods are being deployed.

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get pods -A
```

4.15.1.7.4.1 Manually Attach a DKP CLI Cluster to the Management Cluster

> ⚠ These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

   ```
   echo ${MANAGED_CLUSTER_NAME}
   ```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
   NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

   ```
   kubectl get workspaces -A
   ```

5. Set the WORKSPACE_NAMESPACE environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace-namespace>
   ```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

   ```
   kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
   template='{{.data.value}}{{ "\n"}}'
   ```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

You can also Create a new Azure Cluster Using the DKP UI (see page 775).

4.15.1.7.4.2 Next Step:

Day 2 - Cluster Operations Management (see page 590)

4.16 AKS Install Options

Enterprise Gov Advanced

For an environment that is AKS on the Azure Infrastructure, install options are provided for you in this one location. Remember, there are always more options in the AKS Infrastructure (see page 1343) section under Custom Installation and Additional Infrastructure Tools (see page 1050), but this will get you operative in the most common scenario.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

Below are the supported environment variable combinations: (Refer to this page to check if your OS is a Supported Operating System (see page 67).)

- AKS Install (see page 556)

> In order to install Kommander, you need to have CAPI components, cert-manager, etc on a self-managed cluster. The CAPI components mean you can control the lifecycle of the cluster, and other clusters. However, because AKS is semi-managed by Azure, the AKS clusters are under Azure control and don't have those components. Therefore, Kommander will not be able to be installed and these clusters will be attached to the management cluster.

4.16.1 AKS Install

Enterprise Gov Advanced

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

4.16.1.1 DKP Prerequisites

Before starting the DKP installation, verify that you have:

- A Management cluster with DKP and the Kommander component installed:

An AKS cluster cannot be a Management (see page 80) or Essential (see page 81) cluster. To install DKP on your AKS cluster, first ensure you have a Management cluster with DKP and the Kommander component installed, that handles the lifecycle of your AKS cluster.

- An x86_64-based Linux or macOS machine with a supported version of the operating system (see page 67).
- A Self-managed Azure cluster (see page 1324), if you used the Day 1-Basic Install for Azure instructions, your cluster was created using `--self-managed` flag and therefore is already a self-managed cluster.
- Download the `dkp` binary (see page 76) for Linux, or macOS. To check which version of DKP you installed for compatibility reasons, run the `dkp version -h` command (dkp version (see page 1943)).
- Docker[412] version 18.09.2 or later.
- kubectl[413] for interacting with the running cluster.
- The Azure CLI[414].
- A valid Azure account used to sign in to the Azure CLI[415].
- All Resource requirements (see page 0)

4.16.1.2 AKS Prerequisites

Follow these steps:

1. Log in to Azure:

```
az login
```

```
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "id": "b1234567-abcd-11a1-a0a0-1234a5678b90",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Mesosphere Developer Subscription",
    "state": "Enabled",
    "tenantId": "a1234567-b132-1234-1a11-1234a5678b90",
```

412 https://docs.docker.com/get-docker/
413 https://kubernetes.io/docs/tasks/tools/#kubectl
414 https://docs.microsoft.com/en-us/cli/azure/install-azure-cli
415 https://docs.microsoft.com/en-us/cli/azure/authenticate-azure-cli?view=azure-cli-latest

```
    "user": {
      "name": "user@azuremesosphere.onmicrosoft.com",
      "type": "user"
    }
  }
]
```

2. Create an Azure Service Principal (SP) by running the following command:

   NOTE: If an SP with the name exists, this command will rotate the password.

```
az ad sp create-for-rbac --role contributor --name "$(whoami)-konvoy" --
scopes=/subscriptions/$(az account show --query id -o tsv)
```

```
{
  "appId": "7654321a-1a23-567b-b789-0987b6543a21",
  "displayName": "azure-cli-2021-03-09-23-17-06",
  "password": "Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C",
  "tenant": "a1234567-b132-1234-1a11-1234a5678b90"
}
```

3. Set the required environment variables:

```
export AZURE_SUBSCRIPTION_ID="<id>"       # b1234567-abcd-11a1-
a0a0-1234a5678b90
export AZURE_TENANT_ID="<tenant>"         # a1234567-b132-1234-1a11-1234a5678b9
0
export AZURE_CLIENT_ID="<appId>"          # 7654321a-1a23-567b-
b789-0987b6543a21
export AZURE_CLIENT_SECRET="<password>"   # Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C
```

4. Base64 encode the same environment variables:

```
export AZURE_SUBSCRIPTION_ID_B64="$(echo -n "${AZURE_SUBSCRIPTION_ID}" | base64
| tr -d '\n')"
export AZURE_TENANT_ID_B64="$(echo -n "${AZURE_TENANT_ID}" | base64 | tr -d
'\n')"
export AZURE_CLIENT_ID_B64="$(echo -n "${AZURE_CLIENT_ID}" | base64 | tr -d
'\n')"
export AZURE_CLIENT_SECRET_B64="$(echo -n "${AZURE_CLIENT_SECRET}" | base64 |
tr -d '\n')"
```

5. Check to see what version of Kubernetes is available in your region. When deploying with AKS, you must pick a version of Kubernetes that is available in AKS and use that version for subsequent steps.

To find out the list of available Kubernetes versions in the Azure Region you are using, run the following command, substituting `<your-location>` for the Azure region you're deploying to:

```
az aks get-versions -o table --location <your-location>
```

The output from this command resembles the following:

```
az aks get-versions -o table --location westus
KubernetesVersion    Upgrades
-------------------  ----------------------------------------
1.27.6(preview)      None available
1.27.3(preview)      1.27.6(preview)
1.27.1(preview)      1.27.3(preview)
1.26.6               1.27.1(preview), 1.27.3(preview)
1.26.3               1.26.6, 1.27.1(preview), 1.27.3(preview)
1.25.11              1.26.3, 1.26.6
1.25.6               1.25.11, 1.26.3, 1.26.6
1.24.15              1.25.6, 1.25.11
1.24.10              1.24.15, 1.25.6, 1.25.11
```

6. Choose a version of Kubernetes to install from the list of `KubernetesVersion` , choosing a compatible version as documented in the Supported Kubernetes Versions (see page 1706) for this version of DKP. The version listed in the command is an example:

   ```
   export KUBERNETES_VERSION=1.28.7
   ```

4.16.1.3 Next Step:

AKS: Create Image (see page 559)

4.16.1.4 AKS: Create Image

Enterprise Gov Advanced

AKS best practices discourage building custom images. If the image is customized, it breaks some of the autoscaling and security capabilities of AKS. Since custom virtual machine images are discouraged in AKS, Konvoy Image Builder (KIB) does not include any support for building custom machine images for AKS.

4.16.1.4.1 Next Step:

AKS: Cluster Creation (see page 560)

4.16.1.5 AKS: Create an AKS Cluster

Enterprise Gov Advanced

When creating a Managed (see page 80) cluster on AKS infrastructure, you can choose from multiple configuration types. Create a new Kubernetes cluster in a non-air-gapped environment with the steps below.

4.16.1.5.1 Use DKP to create a new AKS cluster

Ensure that the `KUBECONFIG` environment variable is set to the Management cluster by running `export KUBECONFIG=<Management_cluster_kubeconfig>.conf`.

4.16.1.5.2 Name Your Cluster

Give your cluster a unique name suitable for your environment.

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[416] for more naming information.

4.16.1.5.3 Create a New AKS Kubernetes Cluster

1. Set the environment variable to a name for this cluster.

```
export CLUSTER_NAME=<aks-example>
```

2. Check to see what version of Kubernetes is available in your region. When deploying with AKS, you need to declare the version of Kubernetes you wish to use by running the following command, substituting `<your-location>` for the Azure region you're deploying to:

```
az aks get-versions -o table --location <your-location>
```

416 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

3. Set the version of Kubernetes you've chosen:
   NOTE: Refer to the current release Kubernetes compatibility table (see page 139) for correct version to use and choose an available 1.27.x version. The version listed in the command is an example.

   ```
   export KUBERNETES_VERSION=1.27.6
   ```

4. Create the cluster:

   ```
   dkp create cluster aks --cluster-name=${CLUSTER_NAME} --additional-tags=owner=$
   (whoami) --kubernetes-version=${KUBERNETES_VERSION}
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

   ```
   Generating cluster resources
   cluster.cluster.x-k8s.io/aks-example created
   azuremanagedcontrolplane.infrastructure.cluster.x-k8s.io/aks-example created
   azuremanagedcluster.infrastructure.cluster.x-k8s.io/aks-example created
   machinepool.cluster.x-k8s.io/aks-example created
   azuremanagedmachinepool.infrastructure.cluster.x-k8s.io/cp6dsz8 created
   machinepool.cluster.x-k8s.io/aks-example-md-0 created
   azuremanagedmachinepool.infrastructure.cluster.x-k8s.io/mp6gglj created
   clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-aks-example
   created
   configmap/cluster-autoscaler-aks-example created
   clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-aks-example
   created
   configmap/node-feature-discovery-aks-example created
   clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-aks-example
   created
   configmap/nvidia-feature-discovery-aks-example created
   ```

4.16.1.5.4 Inspecting or Editing the Cluster Objects

Use your favorite editor.

NOTE: Editing the cluster objects requires some understanding of Cluster API. Edits can prevent the cluster from deploying successfully.

The objects are Custom Resources[417] defined by Cluster API components, and they belong in three different categories:

- Cluster
  A *Cluster* object has references to the infrastructure-specific and control plane objects.
- Control Plane
- Node Pool
  A Node Pool is a collection of machines with identical properties. For example, a cluster might have one Node Pool with large memory capacity, another Node Pool with GPU support. Each Node Pool is described by three objects: The MachinePool references an object that describes the configuration of Kubernetes components (for example, kubelet) deployed on each node pool machine, and an infrastructure-specific object that describes the properties of all node pool machines. Here, it references a *KubeadmConfigTemplate*.

For in-depth documentation about the objects, read Concepts[418] in the Cluster API Book.

1. Wait for the cluster control-plane to be ready:

   ```
   kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
   timeout=20m
   ```

   ```
   cluster.cluster.x-k8s.io/aks-example condition met
   ```

   The `READY` status will become `True` after the cluster control-plane becomes ready.

2. After the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. DKP provides a command to describe the current status of the cluster:

   ```
   dkp describe cluster -c ${CLUSTER_NAME}
   ```

   ```
   NAME                                                    READY  SEVERITY
   REASON  SINCE  MESSAGE
   Cluster/aks-example                                     True
   48m
   ├─ClusterInfrastructure - AzureManagedCluster/aks-example
   └─ControlPlane - AzureManagedControlPlane/aks-example
   ```

3. As they progress, the controllers also create Events. List the Events using this command:

417 https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/
418 https://cluster-api.sigs.k8s.io/user/concepts.html

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, the example uses `grep`. It is also possible to use separate commands to get Events for specific objects. For example, `kubectl get events --field-selector involvedObject.kind="AKSCluster"` and `kubectl get events --field-selector involvedObject.kind="AKSMachine"`.

```
48m         Normal    SuccessfulSetNodeRefs               machinepool/aks-
example-md-0                 [{Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000000 UID:e3c30389-660d-46f5-b9d7-219f80b5674d APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000001 UID:300d71a0-f3a7-4c29-9ff1-1995ffb9cfd3 APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000002 UID:8eae2b39-a415-425d-8417-d915a0b2fa52 APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000003 UID:3e860b88-f1a4-44d1-b674-a54fad599a9d APIVersion:
ResourceVersion: FieldPath:}]
6m4s        Normal    AzureManagedControlPlane available
azuremanagedcontrolplane/aks-example          successfully reconciled
48m         Normal    SuccessfulSetNodeRefs               machinepool/aks-
example                      [{Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000000 UID:e3c30389-660d-46f5-b9d7-219f80b5674d APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000001 UID:300d71a0-f3a7-4c29-9ff1-1995ffb9cfd3 APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000002 UID:8eae2b39-a415-425d-8417-d915a0b2fa52 APIVersion:
ResourceVersion: FieldPath:}]
```

4.16.1.5.5 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

More information about AKS can be found in the AKS Infrastructure (see page 1343) under Custom Installation and Additional Infrastructure Tools (see page 1050).

4.16.1.5.6 Next Steps:

AKS: Retrieve kubeconfig for AKS Cluster (see page 563)

4.16.1.6 AKS: Retrieve kubeconfig for AKS Cluster

Enterprise Gov Advanced

4.16.1.6.1 Learn to interact with your AKS Kubernetes cluster

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster.

Before you start, make sure you have created a workload cluster, as described in AKS: Create an AKS Cluster (see page 560) .

4.16.1.6.2 Explore the New AKS Cluster

1. Get a kubeconfig file for the workload cluster:

   When the workload cluster is created, the cluster lifecycle services generate a kubeconfig file for the workload cluster, and write it to a *Secret*. The kubeconfig file is scoped to the cluster administrator.

   Get the kubeconfig from the *Secret*, and write it to a file, using this command:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

2. List the Nodes using this command:

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
   ```

   ```
   NAME                                STATUS   ROLES   AGE   VERSION
   aks-cp6dsz8-41174201-vmss000000   Ready    agent   56m   v1.28.7
   aks-cp6dsz8-41174201-vmss000001   Ready    agent   55m   v1.28.7
   aks-cp6dsz8-41174201-vmss000002   Ready    agent   56m   v1.28.7
   aks-mp6gglj-41174201-vmss000000   Ready    agent   55m   v1.28.7
   aks-mp6gglj-41174201-vmss000001   Ready    agent   55m   v1.28.7
   aks-mp6gglj-41174201-vmss000002   Ready    agent   55m   v1.28.7
   aks-mp6gglj-41174201-vmss000003   Ready    agent   56m   v1.28.7
   ```

NOTE: It may take a few minutes for the Status to move to `Ready` while the Pod network is deployed. The Nodes' Status should change to Ready soon after the `calico-node` DaemonSet Pods are Ready.

3. List the Pods using this command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get --all-namespaces pods
```

```
NAMESPACE                NAME                                                READY
STATUS     RESTARTS   AGE
calico-system            calico-kube-controllers-5dcd4b47b5-tgslm            1/1
Running    0          3m58s
calico-system            calico-node-46dj9                                   1/1
Running    0          3m58s
calico-system            calico-node-crdgc                                   1/1
Running    0          3m58s
calico-system            calico-node-m7s7x                                   1/1
Running    0          3m58s
calico-system            calico-node-qfkqc                                   1/1
Running    0          3m57s
calico-system            calico-node-sfqfm                                   1/1
Running    0          3m57s
calico-system            calico-node-sn67x                                   1/1
Running    0          3m53s
calico-system            calico-node-w2pvt                                   1/1
Running    0          3m58s
calico-system            calico-typha-6f7f59969c-5z4t5                       1/1
Running    0          3m51s
calico-system            calico-typha-6f7f59969c-ddzqb                       1/1
Running    0          3m58s
calico-system            calico-typha-6f7f59969c-rr4lj                       1/1
Running    0          3m51s
kube-system              azure-ip-masq-agent-4f4v6                           1/1
Running    0          4m11s
kube-system              azure-ip-masq-agent-5xfh2                           1/1
Running    0          4m11s
kube-system              azure-ip-masq-agent-9hlk8                           1/1
Running    0          4m8s
kube-system              azure-ip-masq-agent-9vsgg                           1/1
Running    0          4m16s
kube-system              azure-ip-masq-agent-b9wjj                           1/1
Running    0          3m57s
kube-system              azure-ip-masq-agent-kpjtl                           1/1
Running    0          3m53s
kube-system              azure-ip-masq-agent-vr7hd                           1/1
Running    0          3m57s
kube-system              cluster-autoscaler-b4789f4bf-qkfk2                  0/1
Init:0/1   0          3m28s
kube-system              coredns-845757d86-9jf8b                             1/1
Running    0          5m29s
kube-system              coredns-845757d86-h4xfs                             1/1
Running    0          4m
kube-system              coredns-autoscaler-5f85dc856b-xjb5z                 1/1
Running    0          5m23s
```

```
kube-system              csi-azuredisk-node-4n4fx                          3/3
Running    0          3m53s
kube-system              csi-azuredisk-node-8pnjj                          3/3
Running    0          3m57s
kube-system              csi-azuredisk-node-sbt6r                          3/3
Running    0          3m57s
kube-system              csi-azuredisk-node-v25wc                          3/3
Running    0          4m16s
kube-system              csi-azuredisk-node-vfbxg                          3/3
Running    0          4m11s
kube-system              csi-azuredisk-node-w5ff5                          3/3
Running    0          4m11s
kube-system              csi-azuredisk-node-zzgqx                          3/3
Running    0          4m8s
kube-system              csi-azurefile-node-2rpcc                          3/3
Running    0          3m57s
kube-system              csi-azurefile-node-4gqkf                          3/3
Running    0          4m11s
kube-system              csi-azurefile-node-f6k8m                          3/3
Running    0          4m16s
kube-system              csi-azurefile-node-k72xq                          3/3
Running    0          4m8s
kube-system              csi-azurefile-node-vx7r4                          3/3
Running    0          3m53s
kube-system              csi-azurefile-node-zc8kr                          3/3
Running    0          4m11s
kube-system              csi-azurefile-node-zkl6b                          3/3
Running    0          3m57s
kube-system              kube-proxy-4fpb6                                  1/1
Running    0          3m53s
kube-system              kube-proxy-6qfbf                                  1/1
Running    0          4m16s
kube-system              kube-proxy-6wnt2                                  1/1
Running    0          4m8s
kube-system              kube-proxy-cspd5                                  1/1
Running    0          3m57s
kube-system              kube-proxy-nsgq6                                  1/1
Running    0          4m11s
kube-system              kube-proxy-qz2st                                  1/1
Running    0          4m11s
kube-system              kube-proxy-zvh9k                                  1/1
Running    0          3m57s
kube-system              metrics-server-6bc97b47f7-ltkkj                   1/1
Running    0          5m28s
kube-system              tunnelfront-77d68f78bf-t78ck                      1/1
Running    0          5m23s
node-feature-discovery   node-feature-discovery-master-65dc499cd-fxwb5     1/1
Running    0          3m28s
node-feature-discovery   node-feature-discovery-worker-277xc               1/1
Running    0          3m28s
node-feature-discovery   node-feature-discovery-worker-4dq5k               1/1
Running    0          3m28s
```

```
node-feature-discovery   node-feature-discovery-worker-57nb8              1/1
Running    0          3m28s
node-feature-discovery   node-feature-discovery-worker-b4lkl              1/1
Running    0          3m28s
node-feature-discovery   node-feature-discovery-worker-kslst              1/1
Running    0          3m28s
node-feature-discovery   node-feature-discovery-worker-ppjtm              1/1
Running    0          3m28s
node-feature-discovery   node-feature-discovery-worker-x5bgf              1/1
Running    0          3m28s
tigera-operator          tigera-operator-74c4d9cf84-k7css                 1/1
Running    0          5m25s
```

4.16.1.6.3 Next Step:

AKS: Attach Cluster (see page 567)

4.16.1.7 AKS: Attach Cluster

ENTERPRISE

4.16.1.7.0.1 Attach an existing AKS cluster

You can attach existing Kubernetes clusters to the Management Cluster. After attaching the cluster, you can use the UI to examine and manage (see page 774) this cluster. The following procedure shows how to attach an existing Azure Kubernetes Service (AKS) cluster.

4.16.1.7.1 Before you Begin

This procedure requires the following items and configurations:

- A fully configured and running Azure AKS[419]cluster with administrative privileges.
- The current version DKP Enterprise is installed (see page 146)on your cluster.
- Ensure you have installed `kubectl` in your Management cluster.

> This procedure assumes you have an existing and spun up Azure AKS cluster(s) with administrative privileges. Refer to the Azure AKS[420]for setup and configuration information.

419 https://azure.microsoft.com/en-us/products/kubernetes-service/
420 https://azure.microsoft.com/en-us/products/kubernetes-service/

4.16.1.7.2 Attach AKS Clusters

Ensure that the `KUBECONFIG` environment variable is set to the Management cluster before attaching by running:

```
export KUBECONFIG=<Management_cluster_kubeconfig>.conf
```

4.16.1.7.2.1 Ensure you have access to your AKS clusters

1. Ensure you are connected to your AKS clusters. Enter the following commands for each of your clusters:

   ```
   kubectl config get-contexts
   kubectl config use-context <context for first AKS cluster>
   ```

2. Confirm `kubectl` can access the AKS cluster:

   ```
   kubectl get nodes
   ```

4.16.1.7.2.2 Create a kubeconfig file for your AKS cluster

To get started, ensure you have kubectl[421] set up and configured with ClusterAdmin[422] for the cluster you want to connect to Kommander.

1. Create the necessary service account:

   ```
   kubectl -n kube-system create serviceaccount kommander-cluster-admin
   ```

2. Create a token secret for the `serviceaccount`:

   ```
   kubectl -n kube-system create  -f - <<EOF
   apiVersion: v1
   kind: Secret
   metadata:
     name: kommander-cluster-admin-sa-token
     annotations:
       kubernetes.io/service-account.name: kommander-cluster-admin
   type: kubernetes.io/service-account-token
   ```

421 https://kubernetes.io/docs/tasks/tools/#kubectl
422 https://kubernetes.io/docs/concepts/cluster-administration/

```
EOF
```

3. Verify that the `serviceaccount` token is ready by running this command:

   ```
   kubectl -n kube-system get secret kommander-cluster-admin-sa-token -oyaml
   ```

   Verify that the `data.token` field is populated. The output should be similar to this:

   ```
   apiVersion: v1
   data:
     ca.crt: LS0tLS1CRUdJTiBDR...
     namespace: ZGVmYXVsdA==
     token: ZXlKaGJHY2lPaUpTVX...
   kind: Secret
   metadata:
     annotations:
       kubernetes.io/service-account.name: kommander-cluster-admin
       kubernetes.io/service-account.uid: b62bc32e-b502-4654-921d-94a742e273a8
     creationTimestamp: "2022-08-19T13:36:42Z"
     name: kommander-cluster-admin-sa-token
     namespace: default
     resourceVersion: "8554"
     uid: 72c2a4f0-636d-4a70-9f1c-55a75f15e520
   type: kubernetes.io/service-account-token
   ```

4. Configure the new service account for `cluster-admin` permissions:

   ```
   cat << EOF | kubectl apply -f -
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRoleBinding
   metadata:
     name: kommander-cluster-admin
   roleRef:
     apiGroup: rbac.authorization.k8s.io
     kind: ClusterRole
     name: cluster-admin
   subjects:
   - kind: ServiceAccount
     name: kommander-cluster-admin
     namespace: kube-system
   EOF
   ```

5. Set up the following environment variables with the access data that is needed for producing a new kubeconfig file:

```
export USER_TOKEN_VALUE=$(kubectl -n kube-system get secret/kommander-cluster-
admin-sa-token -o=go-template='{{.data.token}}' | base64 --decode)
export CURRENT_CONTEXT=$(kubectl config current-context)
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-
template='{{range .contexts}}{{if eq .name "'''${CURRENT_CONTEXT}'''"}}
{{ index .context "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}"{{with
index .cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{ end }}')
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}
{{ .cluster.server }}{{end}}{{ end }}')
```

6. Confirm these variables have been set correctly:

```
export -p | grep -E 'USER_TOKEN_VALUE|CURRENT_CONTEXT|CURRENT_CLUSTER|
CLUSTER_CA|CLUSTER_SERVER'
```

7. Generate a kubeconfig file that uses the environment variable values from the previous step:

```
cat << EOF > kommander-cluster-admin-config
apiVersion: v1
kind: Config
current-context: ${CURRENT_CONTEXT}
contexts:
- name: ${CURRENT_CONTEXT}
  context:
    cluster: ${CURRENT_CONTEXT}
    user: kommander-cluster-admin
    namespace: kube-system
clusters:
- name: ${CURRENT_CONTEXT}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
users:
- name: kommander-cluster-admin
  user:
    token: ${USER_TOKEN_VALUE}
EOF
```

8. This process produces a file in your current working directory called `kommander-cluster-admin-config`. The contents of this file are used in Kommander to attach the cluster.
   Before importing this configuration, verify the `kubeconfig` file can access the cluster:

```
kubectl --kubeconfig $(pwd)/kommander-cluster-admin-config get all --all-
namespaces
```

4.16.1.7.2.3 Finalize attaching your cluster from the UI

Now that you have kubeconfig, go to the DKP UI and follow these steps below:

1. From the top menu bar, select your target workspace.
2. On the Dashboard page, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
3. Select **Attach Cluster**.
4. Select the **No additional networking restrictions** card. Alternatively, if you must use network restrictions, stop following the steps below, and see the instructions on the page Attach a cluster WITH network restrictions (see page 804).
5. Upload the kubeconfig file you created in the previous section (or copy its contents) into the **Cluster Configuration** section.
6. The **Cluster Name** field automatically populates with the name of the cluster in the kubeconfig. You can edit this field with the name you want for your cluster.
7. Add labels to classify your cluster as needed.
8. Select **Create** to attach your cluster.

If a cluster has limited resources to deploy all the federated platform services, it will fail to stay attached in the DKP UI. If this happens, ensure your system has sufficient resources for all pods.

Next Steps:
Day 2 - Cluster Operations Management (see page 590)

4.17 GCP Install Options

For an environment that is Google Cloud Platform, an install is provided for you in this one location.

Remember, there are always more options for custom YAML in the GCP Infrastructure (see page 1486) section for Google Cloud, but this will get you operative in the most common scenario.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

Below are the supported environment variable combinations: (Refer to this page to check if your OS is a Supported Operating System (see page 67).)

- GCP Install (see page 572)

Additional Resource Information specific to GCP is below.

- **Control Plane Nodes** - DKP on GCP defaults to deploying an `n2-standard-4` instance with an 80GiB root volume for control plane nodes, which meets the above requirements.
- **Worker Nodes** - DKP on GCP defaults to deploying a `n2-standard-8` instance with an 80GiB root volume for worker nodes, which meets the above requirements.

4.17.1 GCP Install

For an environment that is on the GCP Infrastructure, install options are provided for you in this one location. Remember, there are always more options in the GCP Infrastructure (see page 1486) section under Custom Installation and Additional Infrastructure Tools (see page 1050), but this will get you up and running in the most common scenario.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

4.17.1.1 GCP Prerequisites:

- Verify that your Google Cloud project does not have the Enable OS Login feature enabled.

The Enable OS Login feature is sometimes enabled by default in GCP projects. If the OS login feature is enabled, KIB will not be able to `ssh` to the VM instances it creates and will not be able to successfully create an image.
To check if it is enabled, use the commands on this page https://cloud.google.com/compute/docs/metadata/setting-custom-metadata#console_2 to inspect the metadata configured in in your project. If you find the the `enable-oslogin` flag set to TRUE, you must remove (or set it to FALSE) to use KIB.

- The user creating the Service Accounts needs additional privileges in addition to the Editor role.
  - See GCP Prerequisite Roles (see page 1488)

4.17.1.2 Next Step:

GCP: Create Image (see page 573)

4.17.1.3 GCP: Create Image

This procedure describes how to use the Konvoy Image Builder (see page 1626) (KIB) to create a Cluster API[423] compliant GCP image. GCP images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create a GCP image of your current computer system settings and software. The GCP image can then be replicated and distributed, creating your computer system for other users. KIB uses variable overrides (see page 1670) to specify base image and container images to use in your new GCP image.

> ⚠ Google Cloud Platform does not publish images. You must first build the image using Konvoy Image Builder. Explore the Customize your Image (see page 1661) topic for more options. For more information regarding using the image in creating clusters, refer to the GCP Infrastructure (see page 1486) section of the documentation.

4.17.1.3.1 DKP Prerequisites

Before you begin, you must:

- Download the KIB (see page 0) bundle for your version of DKP.
- Check the Supported Infrastructure Operating Systems (see page 67)
- Check the Supported Kubernetes Version (see page 1706) for your Provider.
- Create a working Docker or other registry setup.
- On Debian-based Linux distributions, install a version of the cri-tools[424] package known to be compatible with both the Kubernetes and container runtime versions.
- Verify that your Google Cloud project does not have the Enable OS Login feature enabled. See below for more information:

> ⚠ The Enable OS Login feature is sometimes enabled by default in GCP projects. If the OS login feature is enabled, KIB will not be able to `ssh` to the VM instances it creates and will not be able to successfully create an image.

423 https://cluster-api.sigs.k8s.io/
424 https://github.com/kubernetes-sigs/cri-tools

To check if it is enabled, use the commands on this page https://cloud.google.com/compute/docs/metadata/setting-custom-metadata#console_2 to inspect the metadata configured in in your project. If you find the the `enable-oslogin` flag set to TRUE, you must remove (or set it to FALSE) to successfully use KIB.

4.17.1.3.2 GCP Prerequisite Roles

If you are creating your image on either a non-GCP instance or one that does not have the required roles (see page 1488), you must either:

- Create a GCP service account[425].
- If you have already created a service account, retrieve the credentials for an existing service account.
- Export the static credentials that will be used to create the cluster:

```
export GCP_B64ENCODED_CREDENTIALS=$(base64 < "$
{GOOGLE_APPLICATION_CREDENTIALS}" | tr -d '\n')
```

Make sure to rotate static credentials for increased security.

4.17.1.3.3 Build the GCP image

1. Run the `konvoy-image` command to build and validate the image:

```
./konvoy-image build gcp --project-id ${GCP_PROJECT} --network ${NETWORK_NAME}
images/gcp/ubuntu-2004.yaml
```

2. KIB will run and print out the name of the created image, you will use this name when creating a Kubernetes cluster. See sample output below:

```
...
==> ubuntu-2004-focal-v20220419: Deleting instance...
    ubuntu-2004-focal-v20220419: Instance has been deleted!
==> ubuntu-2004-focal-v20220419: Creating image...
==> ubuntu-2004-focal-v20220419: Deleting disk...
    ubuntu-2004-focal-v20220419: Disk has been deleted!
==> ubuntu-2004-focal-v20220419: Running post-processor: manifest
Build 'ubuntu-2004-focal-v20220419' finished after 7 minutes 46 seconds.
```

425 https://cloud.google.com/iam/docs/service-account-overview

```
==> Wait completed after 7 minutes 46 seconds

==> Builds finished. The artifacts of successful builds are:
--> ubuntu-2004-focal-v20220419: A disk image was created: konvoy-ubuntu-2004-1-2
3-7-1658523168
--> ubuntu-2004-focal-v20220419: A disk image was created: konvoy-ubuntu-2004-1-2
3-7-1658523168
```

Ensure you have named the correct YAML file for your OS[426] in the `konvoy-image build` command.

3. To find a list of images you have created in your account, run the following command:

```
gcloud compute images list --no-standard-images
```

4.17.1.3.3.1 Next Step:

GCP: Cluster Creation (see page 575)

4.17.1.4 GCP: Create the Management Cluster

Create a new Google Cloud Platform Kubernetes cluster in a non-air-gapped environment with the steps below.

DKP uses the GCP CSI driver as the default storage provider (see page 110). Use a Kubernetes CSI[427] compatible storage that is suitable for production.

426 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/gcp
427 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

4.17.1.4.1 Name your Cluster

1. Give your cluster a unique name suitable for your environment. The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[428] for more naming information.

   In GCP it is critical that the name is unique, as no two clusters in the same GCP account can have the same name.

2. Set the environment variable:

   ```
   export CLUSTER_NAME=<gcp-example>
   ```

To increase Docker Hub's rate limit[429] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username=<username> --registry-mirror-password=<password>` on the `dkp create cluster` command.

4.17.1.4.2 Create a New GCP Cluster

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools is to ensure your cluster has nodes deployed in multiple Availability Zones.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single zone. You may create additional node pools in other zones with the `dkp create nodepool` command. The default region for the availability zones is `us-west1`.

Google Cloud Platform does not publish images. You must first build the image using Konvoy Image Builder (see page 1626).

428 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/
429 https://docs.docker.com/docker-hub/download-rate-limit/

1. Create an image using Konvoy Image Builder (KIB) (see page 1626) and then export the image name:

```
export IMAGE_NAME=projects/${GCP_PROJECT}/global/images/<image_name_from_kib>
```

2. Ensure your subnets (see page 1065) do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

3. **(Optional)** Modify Control Plane Audit logs - Users can make modifications to the `KubeadmControlplane` cluster-api object to configure different `kubelet` options. See the following guide (see page 1613) if you wish to configure your control plane beyond the existing options that are available from flags.

4. **(Optional)** Determine what VPC Network to use. All GCP accounts come with a preconfigured VPC Network named `default`, which will be used if you do not specify a different network.
   To use a different VPC network for your cluster, create one by following these instructions for Create and Manage VPC Networks[430]. Then specify the `--network <new_vpc_network_name>` option on the create cluster command below. More information is available on GCP Cloud Nat[431] and network flag.

5. Create a Kubernetes cluster. The following example shows a common configuration. See dkp create cluster gcp (see page 1869) reference for the full list of cluster creation options.

```
dkp create cluster gcp \
--cluster-name=${CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--project=${GCP_PROJECT} \
--image=${IMAGE_NAME} \
--self-managed
```

430 https://cloud.google.com/vpc/docs/create-modify-vpc-networks#create-auto-network

431 https://github.com/kubernetes-sigs/cluster-api-provider-gcp/blob/3406adaae0f8f65a615844e55d856d306eddacc1/docs/book/src/topics/prerequisites.md#cloud-nat

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

6. Wait for the cluster control-plane to be ready:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=20m
```

7. After the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. Konvoy provides a command to describe the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

Output:

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/gcp-example                                                 True
52s
├─ClusterInfrastructure - GCPCluster/gcp-example
├─ControlPlane - KubeadmControlPlane/gcp-example-control-plane      True
52s
| ├─Machine/gcp-example-control-plane-6fbzn                         True
2m32s
| | └─MachineInfrastructure - GCPMachine/gcp-example-control-plane-62g6s
| ├─Machine/gcp-example-control-plane-jf6s2                         True
7m36s
| | └─MachineInfrastructure - GCPMachine/gcp-example-control-plane-bsr2z
| └─Machine/gcp-example-control-plane-mnbfs                         True
54s
|   └─MachineInfrastructure - GCPMachine/gcp-example-control-plane-s8xsx
└─Workers
  └─MachineDeployment/gcp-example-md-0                              True
78s
    ├─Machine/gcp-example-md-0-68b86fddb8-8glsw                     True
2m49s
    | └─MachineInfrastructure - GCPMachine/gcp-example-md-0-zls8d
    ├─Machine/gcp-example-md-0-68b86fddb8-bvbm7                     True
2m48s
    | └─MachineInfrastructure - GCPMachine/gcp-example-md-0-5zcvc
    ├─Machine/gcp-example-md-0-68b86fddb8-k9499                     True
2m49s
```

```
    │ └─MachineInfrastructure - GCPMachine/gcp-example-md-0-k8h5p
    └─Machine/gcp-example-md-0-68b86fddb8-l6vfb                    True
2m49s
      └─MachineInfrastructure - GCPMachine/gcp-example-md-0-9h5vn
```

- A self-managed cluster refers to one in which the CAPI resources and controllers that describe and manage it are running on the same cluster they are managing. As part of the underlying processing using the `--self-managed` flag, the DKP CLI:
  - creates a bootstrap cluster
  - creates a workload cluster
  - moves CAPI controllers from the bootstrap cluster to the workload cluster, making it self-managed
  - deletes the bootstrap cluster

To understand how this process works step by step, you can find customizable steps in GCP Infrastructure (see page 1486) under Custom Installation and Additional Infrastructure Tools (see page 1050).

4.17.1.4.2.1 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to:

Verify your Cluster and DKP Installation (see page 1622).

4.17.1.4.2.2 Next Steps:

GCP: Install Kommander (see page 579)

4.17.1.5 GCP: Install Kommander

You have installed the Konvoy component and created a cluster. Now it is time to Install Kommander which will allow you to access the UI and attach new or existing clusters to monitor.

4.17.1.5.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

4.17.1.5.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

```
export CLUSTER_NAME=<your-management-cluster-name>
```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} >> ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1558) for the deployment:

```
dkp install kommander --init > kommander.yaml
```

4. *If required:* Customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, Rook Ceph customization (Pre-provisioned envs (see page 1541)), etc.

4.17.1.5.1.2 Enable DKP Catalog Applications and Install Kommander

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.7.0
```

   If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

4.17.1.5.1.3 Next Step:

GCP: Verify Install and Log in the UI (see page 581)

4.17.1.6 GCP: Verify Install and Log in the UI

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

NOTE: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

4.17.1.6.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

Log in to the UI

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

4.17.1.6.2 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

4.17.1.6.3 Next Step:

GCP: Subsequent New Clusters (see page 584)

4.17.1.7 GCP: Create a Managed Cluster Using the DKP CLI

Enterprise Gov Advanced

Use this procedure to create GCP Managed clusters using the DKP command line interface (CLI)

> ⚠ When creating Managed clusters (see page 79), you do not need to create and move CAPI objects, or install the Kommander component. Those tasks are only done on Management clusters (see page 79)!

Your new, managed cluster needs to be part of either a new Workspace or an existing Workspace under a management cluster. Create or locate a workspace name to get started with your managed cluster.

4.17.1.7.1 Make New Managed Cluster Part of a Workspace

1. If you have an existing Workspace name, run this command to find the name:
   ⚠ **NOTE**: If you need to create a new Workspace, follow the instructions to Create a New Workspace (see page 678).

   ```
   kubectl get workspace -A
   ```

2. When you have the Workspace name, set the `WORKSPACE_NAMESPACE` environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

4.17.1.7.2 Name your Cluster

1. Give your cluster a unique name suitable for your environment.

   In GCP it is critical that the name is unique, as no two clusters in the same GCP account can have the same name.

2. Set the environment variable:

   ```
   export MANAGED_CLUSTER_NAME=<gcp-additional>
   ```

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[432] for more naming information.

To increase Docker Hub's rate limit[433], use your Docker Hub credentials when creating the cluster by setting the following flag, `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username=<username> --registry-mirror-password=<password>` on the `dkp create cluster` command.

4.17.1.7.3 Create a New GCP Cluster

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools is to ensure your cluster has nodes deployed in multiple Availability Zones.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single zone. You may create additional node pools in other zones with the `dkp create nodepool` command. The default region for the availability zones is `us-west1`.

Google Cloud Platform does not publish images. You must first build the image using Konvoy Image Builder (see page 1626).

The below instructions tell you how to create a cluster and have it automatically attach to the workspace you set above.

If you do not set a workspace, it will be created in the `default` workspace, and you need to

432 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

433 https://docs.docker.com/docker-hub/download-rate-limit/

take additional steps to attach to a workspace later. For instructions on how to do this, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29920384/Attach+a+Kubernetes+Cluster). (see page 784).

1. Create an image using Konvoy Image Builder (KIB) (see page 1626) and then export the image name:

```
export IMAGE_NAME=projects/${GCP_PROJECT}/global/images/<image_name_from_kib>
```

2. Create a Kubernetes cluster. The following example shows a common configuration. See dkp create cluster gcp (see page 1869) reference for the full list of cluster creation options:

```
dkp create cluster gcp \
--cluster-name=${MANAGED_CLUSTER_NAME} \
--additional-tags=owner=$(whoami) \
--namespace ${WORKSPACE_NAMESPACE} \
--project=${GCP_PROJECT} \
--image=${IMAGE_NAME} \
--kubeconfig=<management-cluster-kubeconfig-path>
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

4.17.1.7.4 Retrieve the kubeconfig and Explore New GCP Cluster

Follow these steps:

1. Fetch the kubeconfig file with the command:

```
dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} --kubeconfig
<management-cluster-kubeconfig-path> -n ${WORKSPACE_NAMESPACE} > $
{MANAGED_CLUSTER_NAME}.conf
```

2. List the nodes with the following command:

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get nodes
```

3. List the pods with the following command:

NOTE: Wait for the Status to move to Ready while `calico-node` pods are being deployed.

```
kubectl --kubeconfig=${MANAGED_CLUSTER_NAME}.conf get pods -A
```

4.17.1.7.4.1 Manually Attach a DKP CLI Cluster to the Management Cluster

These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

   ```
   echo ${MANAGED_CLUSTER_NAME}
   ```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
   NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

   ```
   kubectl get workspaces -A
   ```

5. Set the WORKSPACE_NAMESPACE environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace-namespace>
   ```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

```
kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
template='{{.data.value}}{{ "\n"}}'
```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

4.17.1.7.4.2 Next Step:

Day 2 - Cluster Operations Management (see page 590)

5 Day 2 - Cluster Operations Management

Use these sections to manage your DKP environment.

5.1 Operations

Manage your cluster and deployed applications using platform applications

After you deploy a DKP cluster and the platform applications you want to use, you are ready to begin managing cluster operations and their application workloads to optimize your organization's productivity.

In most cases, a production cluster requires additional advanced configuration tailored for your environment, ongoing maintenance, authentication and authorization, and other common activities. For example, it is important to monitor cluster activity and collect metrics to ensure application performance and response time, evaluate network traffic patterns, manage user access to services, and verify workload distribution and efficiency.

In addition to advanced configurations, you can also control the appearance of your DKP user interface by adding Banners and Footers. There are different options available depending on the DKP level that you license and install.

5.1.1 Access Control

5.1.1.1 Centrally Manage Access Across Clusters

You can centrally define role-based authorization within DKP UI to control resource access on the management cluster and a set, or all, of the target clusters. These resources are similar to Kubernetes RBAC but with crucial differences, and they make it possible to define the roles and role bindings once, and have them federated to clusters within a given scope.

DKP UI has two conceptual groups of resources that are used to manage access control:

- Kommander Roles: control access to resources on the management cluster.
- Cluster Roles: control access to resources on all target clusters in scope.

Use these two groups of resources to manage access control within 3 levels of scope:

Environment Context (selectable through the DKP UI)	**Kommander Roles**	**Cluster Roles**
Global Manages access to the entire environment.	Create ClusterRoles on the management cluster.	Federates ClusterRoles on all target clusters across all workspaces.
Workspace Manages access to clusters in a specific workspace, for example, in the scope of Multi-Tenancy in DKP (see page 712).	Create namespaced Roles on the management cluster in the workspace namespace.	Federates ClusterRoles on all target clusters in the workspace.
Project Manages access for clusters in a specific project, for example, in the scope of Tenant Projects (see page 713).	Create namespaced Roles on the management cluster in the project namespace.	Federates namespaced Roles on all target clusters in the project in the project namespace.

The role bindings (see page 595) for each level and type create `RoleBindings` or `ClusterRoleBindings` on the clusters that apply to each category.

This approach gives you maximum flexibility over who has access to what resources, conveniently mapped to your existing identity providers' claims.

5.1.1.2 Special Limitation for Kommander Roles

In addition to granting a Kommander Role, you must also grant the appropriate DKP role to allow external users and groups into the UI. See RBAC - DKP UI Authorization (see page 596) for details about the built-in DKP

roles. Here are examples of `ClusterRoleBindings` that grant an Identity provider (IdP) group (in this example, the user group “engineering”) admin access to the Kommander routes:

The property for the `subjects.name` varies depending on the context for which you have established an Identity Provider.

- If you have set up an identity provider for **All Workspaces**:
  - For *groups*: configure the `subjects.name` field to `oidc:<IdP_user_group>`. For example, `oidc:engineering`.
  - For *users*: configure the `subjects.name` field to `<user_email>`. For example, `jane.doe@example.com`
- If you have set up an identity provider for a **Specific Workspace**:
  - For *groups*: configure the `subjects.name` field to `oidc:<workspace_ID>:<IdP_user_group>`. For example, `oidc:tenant-z:engineering`.
  - For *users*: configure the `subjects.name` field to `<workspace_ID>:<user_email>`. For example, `tenant-z:jane.doe@example.com`.
    Run `kubectl get workspaces` to obtain a list of all existing workspaces. The `workspace_ID` is listed under the `NAME` column.

```
cat <<EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eng-kommander-dashboard
  labels:
    "workspaces.kommander.mesosphere.io/rbac": ""
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: dkp-kommander-admin
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: Group
    name: oidc:engineering
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
```

```
metadata:
  name: eng-dkp-routes
  labels:
    "workspaces.kommander.mesosphere.io/rbac": ""
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: dkp-admin
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: Group
    name: oidc:engineering
EOF
```

5.1.1.3 Types of Access Control Objects

Kubernetes role-based access control can be controlled with three different object categories: Groups, Roles and Policies, as explained in more detail below.

5.1.1.3.1 Groups

You can map group and user claims made by your configured identity providers to Kommander groups by selecting Administration / Identity providers in the left sidebar in the global workspace level, and then selecting the **Groups** tab.

The syntax for the Identity Provider groups you add to a DKP Group varies depending on the context for which you have established an Identity Provider.

- If you have set up an identity provider globally, for **All Workspaces**:
  - For *groups*: Add an **Identity Provider Group** in the `oidc:<IdP_user_group>` format. For example, `oidc:engineering`.
  - For *users*: Add an **Identity Provider User** in the `<user_email>`. For example, `jane.doe@example.com`.
- If you have set up an identity provider for a **Specific Workspace**:
  - For *groups*: Add an **Identity Provider Group** in the `oidc:<workspace_name>:<IdP_user_group>` format. For example, `oidc:tenant-z:engineering`.
  - For *users*: Add an **Identity Provider User** in the `<workspace_ID>:<user_email>` format. For example, `tenant-z:jane.doe@example.com`.

Run `kubectl get workspaces` to obtain a list of all existing workspaces. The `workspace_ID` is listed under the `NAME` column.

5.1.1.3.2 Roles

`ClusterRoles` are named collections of rules defining which verbs can be applied to which resources.

- Kommander Roles apply specifically to resources on the management cluster.
- Cluster Roles apply to target clusters within their scope at these levels:
  - Global level - this is all target clusters in all workspaces,
  - Workspace level - this is all target clusters in the workspace,
  - Project level - this is all target clusters that have been added to the project.

5.1.1.3.3 Propagate Workspace Roles to Projects

By default, users granted the Kommander Workspace Admin, Edit, or View roles will also be granted the equivalent Kommander Project Admin, Edit, or View role for any project created in the workspace. Other workspace roles are not automatically propagated to the equivalent role for a project in the workspace.

Each workspace has roles defined using `KommanderWorkspaceRole` resources. Automatic propagation is controlled using the annotation `"workspace.kommander.mesosphere.io/sync-to-project": "true"` on a `KommanderWorkspaceRole` resource. You can manage this only by using the CLI.

```
kubectl get kommanderworkspaceroles -n <WORKSPACE_NAMESPACE>
```

```
NAME                        DISPLAY NAME                      AGE
kommander-workspace-admin   Kommander Workspace Admin Role    2m18s
kommander-workspace-edit    Kommander Workspace Edit Role     2m18s
kommander-workspace-view    Kommander Workspace View Role     2m18s
```

To prevent propagation of the `kommander-workspace-view` role, remove this annotation from the `KommanderWorkspaceRole` resource.

```
kubectl annotate kommanderworkspacerole -n <WORKSPACE_NAMESPACE> kommander-workspace-
view workspace.kommander.mesosphere.io/sync-to-project-
```

To enable propagation of the role, add this annotation to the relevant `KommanderWorkspaceRole` resource.

```
kubectl annotate kommanderworkspacerole -n <WORKSPACE_NAMESPACE> kommander-workspace-
view workspace.kommander.mesosphere.io/sync-to-project=true
```

5.1.1.3.4 Special Limitation for Workspace > Project Role Inheritance

When granting users access to a workspace, you must manually grant access to the projects within that workspace. Each project is created with a set of admin/edit/view roles, and you can choose to add an additional `RoleBinding` to each group or user of the workspace for one of these project roles. Usually, these are prefixed `kommander-project-(admin/edit/view)`. Here is an example `RoleBinding` that grants the Kommander Project Admin role access for the project namespace to the engineering group:

```
cat <<EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: workspace-admin-project1-admin
  namespace: <my-project-namespace-xxxxx>
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: <kommander-project-admin-xxxxx>
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: Group
    name: oidc:engineering
EOF
```

5.1.1.3.5 Role Bindings

Kommander role bindings, cluster role bindings, and project role bindings bind a Kommander group to any number of roles. All groups defined in the **Groups** tab will be present at the global, workspace, or project level, and are ready for you to assign roles to them.

5.1.1.4 Related Information

- Project Role Bindings (see page 757)
- Workspace Role Bindings (see page 711)
- DKP UI RBAC Authorization (see page 596)
- Kubernetes RBAC Authorization[434]

434 https://kubernetes.io/docs/reference/access-authn-authz/rbac/

5.1.1.5 Granting Access to Kubernetes and Kommander Resources

5.1.1.5.1 Grant access to Kommander and Kubernetes resources using RBAC

5.1.1.5.2 Granting Access to External Users

Users and groups from an external identity provider initially have no access to kubernetes resources. Privileges must be granted explicitly by interacting with the RBAC API. This section provides some basic examples for general usage. More information about the RBAC API can be found in the Kubernetes documentation[435].

5.1.1.5.2.1 The Basics

Kubernetes does not provide an identity database for standard users. Users and group membership must be provided by a trusted identity provider. In Kubernetes, RBAC policies are additive, which means that a subject (user, group, or service account) is denied access to a resource unless explicitly granted access by a cluster administrator. You can grant access by binding a subject to a role, which grants some level of access to one or more resources. Kubernetes ships with some default roles[436], which aid in creating broad access control policies.

For example, if you want to make `mary@example.com` a cluster administrator, bind her username to the `cluster-admin` default role as follows:

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: mary-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: mary@example.com
EOF
```

435 https://kubernetes.io/docs/reference/access-authn-authz/rbac/
436 https://kubernetes.io/docs/reference/access-authn-authz/rbac/#default-roles-and-role-bindings

If you have configured an Identity Provider for a specific workspace (see page 714), configure the `subjects.name` field to `<workspace_ID>:<user_email>`. For example, `tenant-z:jane.doe@example.com`.

This user now has the highest level of access which can be achieved. Use the `cluster-admin` role and `system:masters` group sparingly.

5.1.1.5.2.2 Restricting a User to Namespace

A more common example would be to grant a user access to a specific namespace, by creating a RoleBinding (RoleBindings are namespaced scoped). For example, to make the user `bob@example.com` a *reader* of the `baz` namespace, bind the user to the `view` role:

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: bob-view
  namespace: baz
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: view
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: bob@example.com
EOF
```

If you have configured an Identity Provider for a specific workspace (see page 714), configure the `subjects.name` field to `<workspace_ID>:<user_email>`. For example, `tenant-z:jane.doe@example.com`.

The user can now perform non-destructive operations targeting resources in the `baz` namespace only.

5.1.1.5.2.3 Groups

If your external identity provider supports group claims, you can also bind groups to roles. To make the `engineering` LDAP group administrators of the `production` namespace bind the group to the `admin` role:

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: engineering-admin
  namespace: production
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: oidc:engineering
EOF
```

One important distinction from adding users is that all external groups are prefixed with `oidc:`, so a group name becomes `oidc:devops`. This prevents collision with locally defined groups.

The property for the `subjects.name` varies depending on the context for which you have established an Identity Provider.

- If you have set up an identity provider for **All Workspaces**:
  - For *groups*: configure the `subjects.name` field to `oidc:<IdP_user_group>`. For example, `oidc:engineering`.
  - For *users*: configure the `subjects.name` field to `<user_email>`. For example, `jane.doe@example.com`
- If you have set up an identity provider for a **Specific Workspace**:
  - For *groups*: configure the `subjects.name` field to `oidc:<workspace_ID>:<IdP_user_group>`. For example, `oidc:tenant-z:engineering`.

- For *users*: configure the `subjects.name` field to `<workspace_ID>:<user_email>`. For example, `tenant-z:jane.doe@example.com`.
  Run `kubectl get workspaces` to obtain a list of all existing workspaces. The `workspace_ID` is listed under the `NAME` column.

5.1.1.5.3 DKP UI Authorization

The DKP UI, and other HTTP applications protected by Kommander forward authentication, are also authorized by the Kubernetes RBAC API. In addition to Kubernetes API resources, it is possible to define rules which map to HTTP URIs and HTTP verbs. Kubernetes RBAC calls these `nonResourceURLs`, Kommander forward authentication uses these rules to grant or deny access to HTTP endpoints.

5.1.1.5.3.1 Default Roles

Roles have been created for granting access to the dashboard and select applications which expose an HTTP server through the ingress controller. The `cluster-admin` role is actually a system role that defines grants permission to all actions (verbs) on any resource; including non-resource URLs. The default dashboard user is bound to this role.

Granting user `admin` privileges on `/dkp/*` grants `admin` privileges to all sub-resources, even if bindings exist for sub-resources with less privileges.

Dashboard	Role	Path	access
*	cluster-admin	*	read, write, delete
kommander	dkp-view	/dkp/*	read
kommander	dkp-edit	/dkp/*	read, write
kommander	dkp-admin	/dkp/*	read, write, delete
kommander-dashboard	dkp-kommander-view	/dkp/kommander/dashboard/*	read

Dashboard	Role	Path	access
kommander-dashboard	dkp-kommander-edit	/dkp/kommander/dashboard/*	read, write
kommander-dashboard	dkp-kommander-admin	/dkp/kommander/dashboard/*	read, write, delete
alertmanager	dkp-kube-prometheus-stack-alertmanager-view	/dkp/alertmanager/*	read
alertmanager	dkp-kube-prometheus-stack-alertmanager-edit	/dkp/alertmanager/*	read, write
alertmanager	dkp-kube-prometheus-stack-alertmanager-admin	/dkp/alertmanager/*	read, write, delete
centralized-grafana	dkp-centralized-grafana-grafana-view	/dkp/kommander/monitoring/grafana/*	read
centralized-grafana	dkp-centralized-grafana-grafana-edit	/dkp/kommander/monitoring/grafana/*	read, write
centralized-grafana	dkp-centralized-grafana-grafana-admin	/dkp/kommander/monitoring/grafana/*	read, write, delete
centralized-kubecost	dkp-centralized-kubecost-view	/dkp/kommander/kubecost/*	read
centralized-kubecost	dkp-centralized-kubecost-edit	/dkp/kommander/kubecost/*	read, write
centralized-kubecost	dkp-centralized-kubecost-admin	/dkp/kommander/kubecost/*	read, write, delete
grafana	dkp-kube-prometheus-stack-grafana-view	/dkp/grafana/*	read
grafana	dkp-kube-prometheus-stack-grafana-edit	/dkp/grafana/*	read, write

Dashboard	Role	Path	access
grafana	dkp-kube-prometheus-stack-grafana-admin	/dkp/grafana/*	read, write, delete
grafana-logging	dkp-grafana-logging-view	/dkp/logging/grafana/*	read
grafana-logging	dkp-grafana-logging-edit	/dkp/logging/grafana/*	read, write
grafana-logging	dkp-grafana-logging-admin	/dkp/logging/grafana/*	read, write, delete
karma	dkp-karma-view	/dkp/kommander/monitoring/karma/*	read
karma	dkp-karma-edit	/dkp/kommander/monitoring/karma/*	read, write
karma	dkp-karma-admin	/dkp/kommander/monitoring/karma/*	read, write, delete
kubernetes-dashboard	dkp-kubernetes-dashboard-view	/dkp/kubernetes/*	read
kubernetes-dashboard	dkp-kubernetes-dashboard-edit	/dkp/kubernetes/*	read, write
kubernetes-dashboard	dkp-kubernetes-dashboard-admin	/dkp/kubernetes/*	read, write, delete
prometheus	dkp-kube-prometheus-stack-prometheus-view	/dkp/prometheus/*	read
prometheus	dkp-kube-prometheus-stack-prometheus-edit	/dkp/prometheus/*	read, write
prometheus	dkp-kube-prometheus-stack-prometheus-admin	/dkp/prometheus/*	read, write, edit

Dashboard	Role	Path	access
traefik	dkp-traefik-view	/dkp/traefik/*	read
traefik	dkp-traefik-edit	/dkp/traefik/*	read, edit
traefik	dkp-traefik-admin	/dkp/traefik/*	read, edit, delete
thanos	dkp-thanos-query-view	/dkp/kommander/monitoring/query/*	read
thanos	dkp-thanos-query-edit	/dkp/kommander/monitoring/query/*	read, write
thanos	dkp-thanos-query-admin	/dkp/kommander/monitoring/query/*	read, write, delete

This section provides a few examples of binding subjects to the default roles defined for the DKP UI endpoints.

5.1.1.5.3.2 Examples

User

To grant the user `mary@example.com` administrative access to all Kommander resources, bind the user to the `dkp-admin` role:

```
cat << EOF | kubectl apply -f -
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dkp-admin-mary
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: dkp-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: mary@example.com
EOF
```

If you inspect the role, you see what access is now granted:

```
kubectl describe clusterroles dkp-admin
```

```
Name:         dkp-admin
Labels:       app.kubernetes.io/instance=kommander
              app.kubernetes.io/managed-by=Helm
              app.kubernetes.io/version=v2.0.0
              helm.toolkit.fluxcd.io/name=kommander
              helm.toolkit.fluxcd.io/namespace=kommander
              rbac.authorization.k8s.io/aggregate-to-admin=true
Annotations:  meta.helm.sh/release-name: kommander
              meta.helm.sh/release-namespace: kommander
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
  ---------  -----------------  --------------  -----
             [/dkp/*]           []              [delete]
             [/dkp]             []              [delete]
             [/dkp/*]           []              [get]
             [/dkp]             []              [get]
             [/dkp/*]           []              [head]
             [/dkp]             []              [head]
             [/dkp/*]           []              [post]
             [/dkp]             []              [post]
             [/dkp/*]           []              [put]
             [/dkp]             []              [put]
```

The user can now use the HTTP verbs HEAD, GET, DELETE, POST, and PUT when accessing any URL at or under `/dkp`. Provided the downstream application follows REST conventions, this effectively allows read, edit, and delete privileges.

NOTE: To allow users to access the DKP UI, ensure they have the appropriate `dkp-kommander-` role in addition to the Kommander roles granted in the DKP UI. For more information, see the Access Control section of the Kommander documentation (see page 591).

Group

In order to grant view access to the `/dkp/*` endpoints and edit access to the grafana logging endpoint to group `logging-ops`, create the following ClusterRoleBindings:

```
cat << EOF | kubectl apply -f -
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dkp-view-logging-ops
```

```
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: dkp-view
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: oidc:logging-ops
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dkp-logging-edit-logging-ops
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: dkp-logging-edit
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: oidc:logging-ops
EOF
```

Note: external groups must be prefixed by `oidc:`

- The property for the `subjects.name` varies depending on the context for which you have established an Identity Provider.
  - If you have set up an identity provider for **All Workspaces**:
    - For *groups*: configure the `subjects.name` field to `oidc:<IdP_user_group>`. For example, `oidc:engineering`.
    - For *users*: configure the `subjects.name` field to `<user_email>`. For example, `jane.doe@example.com`
  - If you have set up an identity provider for a **Specific Workspace**:
    - For *groups*: configure the `subjects.name` field to `oidc:<workspace_ID>:<IdP_user_group>`. For example, `oidc:tenant-z:engineering`.

- For *users*: configure the `subjects.name` field to `<workspace_ID>:<user_email>`. For example, `tenant-z:jane.doe@example.com`.
  Run `kubectl get workspaces` to obtain a list of all existing workspaces. The `workspace_ID` is listed under the `NAME` column.

Members of `logging-ops` are now able to `view` all resources under `/dkp` and edit all resources under `/dkp/logging/grafana`.

5.1.1.5.3.3 Custom Roles

In the event that one of the predefined roles from DKP does not include all the permissions you need, you can create a custom role. An example set of steps is listed below, but you choose which actions and permissions you want to grant.

1. Select **Access Control** in the **Administration** section of the sidebar menu.
2. Select the **Cluster Roles** tab, and then select the **+ Create Role** button.
3. Give the role a descriptive name, and ensure that **Cluster Role** is selected as the type.
4. For example, to configure a read-only role, select **+ Add Rule**:
   a. select **All Resource Types** in the **Resources** input
   b. select the **get**, **list**, and **watch** verbs with their respective checkboxes
   c. Select **Save**.

Now you can assign your newly-created role to the developers group.

5.1.1.5.4 Accessing the Kubernetes Dashboard

The Kubernetes dashboard offloads authorization directly to the Kubernetes API server. Once authenticated, all users may access the dashboard at `/dkp/kubernetes/` without needing a `dkp` role. However, access to the underlying kubernetes resources exposed by the dashboard are protected by the cluster RBAC policy.

5.1.1.5.5 Further Reading

This page has provides some basic examples of operations which provide the building blocks of creating an access control policy. For more information about creating your own roles and advanced policies, we highly recommend reading the Kubernetes RBAC documentation[437].

For information on how you can add a user to a cluster as an admin, refer to Onboarding Users onto a DKP Cluster (see page 606).

437 https://kubernetes.io/docs/reference/access-authn-authz/rbac/

5.1.1.6 Onboarding Users onto a DKP Cluster

After you install DKP and create a cluster, use this procedure to add new users to your environment.

5.1.1.6.1 Prerequisites

- A Valid DKP License (Essential or Enterprise)
- A running Cluster

5.1.1.6.2 Onboarding a New User to a DKP Cluster

This procedure assumes you are using a LDAP Connector and that you are a Cluster Admin. For information on how you add users using other types of connectors, refer to the following pages:

- OIDC[438]
- SAML[439]
- GitHub[440]

1. Create an LDAP Connector definition and name it `ldap.yaml`. An example is shown below:

```
apiVersion: v1
kind: Secret
metadata:
  name: ldap-password
  namespace: kommander
type: Opaque
stringData:
  password: superSecret
---
apiVersion: dex.mesosphere.io/v1alpha1
kind: Connector
metadata:
  name: ldap
  namespace: kommander
spec:
  enabled: true
```

438 https://dexidp.io/docs/connectors/oidc/
439 https://dexidp.io/docs/connectors/saml/
440 https://dexidp.io/docs/connectors/github/

```
  type: ldap
  displayName: LDAP Test Connector
  ldap:
    host: ldapdce.testdomain
    insecureNoSSL: true
    bindDN: cn=ldapconnector,cn=testgroup,ou=testorg,dc=testdomain
    bindSecretRef:
      name: ldap-password
    userSearch:
      baseDN: dc=testdomain
      filter: "(objectClass=inetOrgPerson)"
      username: uid
      idAttr: uid
      emailAttr: uid
    groupSearch:
      baseDN: ou=testorg,dc=testdomain
      filter: "(objectClass=posixGroup)"
      userMatchers:
      - userAttr: uid
        groupAttr: memberUid
      nameAttr: cn
```

2. Add the connector using the following command:

```
kubectl apply -f ldap.yaml
```

   a. The output should look similar to the following:

```
secret/ldap-password created
connector.dex.mesosphere.io/ldap created
```

3. Add the appropriate role bindings and name the file `new_user.yaml`. Examples for both Single User and Group Bindings are shown below:

The property for the `subjects.name` varies depending on the context for which you have established an Identity Provider.

- If you have set up an identity provider for **All Workspaces**:
  - For *groups*: configure the `subjects.name` field to `oidc:<IdP_user_group>`. For example, `oidc:engineering`.
  - For *users*: configure the `subjects.name` field to `<user_email>`. For example, `jane.doe@example.com`
- If you have set up an identity provider for a **Specific Workspace**:

- For *groups*: configure the `subjects.name` field to `oidc:<workspace_ID>:<IdP_user_group>`. For example, `oidc:tenant-z:engineering`.
- For *users*: configure the `subjects.name` field to `<workspace_ID>:<user_email>`. For example, `tenant-z:jane.doe@example.com`.
  Run `kubectl get workspaces` to obtain a list of all existing workspaces. The `workspace_ID` is listed under the `NAME` column.

- For Single Users:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: newUser
```

- Group Binding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: cluster-admin
  namespace: ml
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: oidc:kommanderAdmins
```

4. Add the role binding(s) using the following command:

```
kubectl apply -f new_user.yaml
```

ClusterRoleBindings permissions are applicable on a global level.
RoleBindings permissions are applicable on a namespace level.

This procedure assumes that the user being added is an admin.
For additional information about the other roles in DKP and their permissions, refer to Granting Access to Kubernetes and Kommander Resources (see page 596)

5.1.2 Identity Providers

Grant access to users in your organization.

DKP supports GitHub (see page 611), LDAP (see page 613), SAML and standard OIDC identity providers such as Google. These identity management providers support the login and authentication process for DKP and your Kubernetes clusters.

You can configure as many identity providers as you want, and users can select from any method when logging in. If you have multiple workspaces in your environment, you can use a single identity provider to manage access to all of them, or choose to configure an identity provider per workspace.

Configuring a dedicated identity provider per workspace can be useful if you want to keep access to your workspaces separate. In this case, users of a specific workspace will have a dedicated login page with the identity provider options configured for their workspace. This setup is particularly helpful if you have multiple tenants (see page 712).

5.1.2.1 Benefits of Using an External Identity Provider

Using an external identity provider is beneficial for many reasons:

- Centralized management of multiple users, to multiple clusters
- Centralized management of password rotation, expiration, etc.
- Support of 2-factor-authentication methods for increased security
- Separate storage of user credentials

5.1.2.2 Prerequisites

To get started with DKP, you must:

- Log in to the UI (see page 1556)

5.1.2.2.1 Limit Access

- The GitHub (see page 611) provider allows you to specify any of the organizations and teams are eligible for access.
- The LDAP (see page 613) provider allows you to configure search filters for either users or groups.
- The OIDC provider cannot limit users based on identity.
- The SAML provider allows users to log in using a single sign-on (SSO) profile.

5.1.2.2.2 Configure an Identity Provider via the UI

1. From the drop-down menu, select the **Global** workspace.
2. Select **Administration > Identity Providers**.
3. Select the **Identity Providers** tab.
4. Select **+ Add Identity Provider**.
5. Select an identity provider.
6. Select the **target workspace** for the identity provider and complete the form field with the relevant details.

> You can configure an identity provider globally, for your entire organization (**All Workspaces** option), or per workspace, enabling multi-tenancy. See How do I enable multi-tenancy? (see page 714) for more information.

7. Select **Save** to create your Identity Provider.

5.1.2.2.3 Temporarily Disabling a Provider

Select the three-dot button on the Identity Providers table and select **Disable** from the drop-down menu. The provider option no longer appears on the login screen.

5.1.2.3 Groups

Enterprise Gov Advanced

Access control groups are configured in the Groups tab of the Identity Providers page. See Access Control (see page 591) for an overview of groups in DKP.

5.1.2.4 Configure GitHub Identity Provider

Enterprise Gov Advanced

Configure GitHub as an identity provider and grant access to DKP.

5.1.2.4.1 Authorize Access to Your Clusters using your GitHub Account

DKP allows configuring access to the UI with GitHub credentials, but it must be configured in the dashboard. To ensure every developer in your GitHub organization has access to your Kubernetes clusters using their GitHub credentials, you will add that option for log in by adding an Identity Provider with information from your GitHub profile in the OAuth application settings.

5.1.2.4.2 Add Identity Provider

To authorize all developers to have access to your clusters using their GitHub credentials, follow these steps to set up GitHub as an identity provider log in option:

1. Start by creating a new OAuth Application in our GitHub Organization by filling out this form[441]. Use your cluster URL in the HomePage URL field.

> **Important:** Use your cluster URL followed by `/dex/callback` as the **Authorization callback URL** by adding to the end of your URL.

2. After you create the application, you will be taken to a settings page. You will need the **Client ID** and **Client Secret** from this page for the DKP UI. If you do not already have a Client Secret for the application, select the **Generate a new client secret** button.
3. Log in to your DKP UI, and from the top menu bar, select the **Global** workspace.
4. Select **Identity Providers** in the **Administration** section of the sidebar menu.
5. Select the **Identity Providers** tab, and then select the **+ Add Identity Provider** button.
6. Ensure GitHub is selected as the identity provider type, and select the target workspace.
7. Copy the **Client ID** and **Client Secret** values from GitHub into this form.
8. Select the checkbox next to **Load All Groups.** Checking the **Load All Groups** checkbox configures `dex` to load all groups configured in the users GitHub identity. This allows you to configure group-specific access to DKP and Kubernetes resources.

441 https://github.com/settings/applications/new

Do not select the checkbox for **Enable Device Flow** before selecting <Register the Application> button.

9. Select **Save** to create your Identity Provider.

5.1.2.4.2.1 Map the Identity Provider Groups to the Kubernetes Groups

1. In the DKP UI, select the **Groups** tab from the **Identity Provider** screen, and then select the **Create Group** button.
2. Give your group a descriptive name in the **Enter Name** box.

The syntax for the Identity Provider groups you add to a DKP Group varies depending on the context for which you have established an Identity Provider.

- If you have set up an identity provider globally, for **All Workspaces**:
  - For *groups*: Add an **Identity Provider Group** in the `oidc:<IdP_user_group>` format. For example, `oidc:engineering`.
  - For *users*: Add an **Identity Provider User** in the `<user_email>`. For example, `jane.doe@example.com`.
- If you have set up an identity provider for a **Specific Workspace**:
  - For *groups*: Add an **Identity Provider Group** in the `oidc:<workspace_name>:<IdP_user_group>` format. For example, `oidc:tenant-z:engineering`.
  - For *users*: Add an **Identity Provider User** in the `<workspace_ID>:<user_email>` format. For example, `tenant-z:jane.doe@example.com`.

    Run `kubectl get workspaces` to obtain a list of all existing workspaces. The `workspace_ID` is listed under the `NAME` column.

3. Add the groups/teams from your GitHub provider under **Identity Provider Groups**. For more information on finding the teams to which you are assigned in GitHub, refer to the GitHub Documentation for GitHub Teams.[442]
4. Click **Save** to create the group, which creates it on the management cluster and federates it to all target clusters.

5.1.2.4.2.2 Assign the Role to the Developers Group

Now that you have a **Group** defined, you can bind one or more **Roles** to this Group. In the following example, we show how to bind the group to the *View Only* role.

1. In the DKP UI, select **Global**, or the target workspace from the top menu bar.
2. Select the **Cluster Role Bindings** tab, and then select the **Add roles** button for your group.
3. Select *View Only* role from the **Roles** drop-down and select <Save>.
4. Follow the example in the Granting Access to Kubernetes and Kommander Resources (see page 596) to grant users access to Kommander paths on your cluster. At a minimum, add a read only path for access to all the Kommander Dashboard views:

Dashboard	Role	Path	Access
kommander-dashboard	dkp-kommander-view	**/dkp/kommander/ dashboard/***	read

When you check your attached clusters and login as a user from your matched groups, you can see every resource, but neither delete or edit them, as intended.

5.1.2.4.2.3 Future Log In

The first login will require you to authorize the GitHub account, so the administrator of the cluster will select <***Authorize github-username***> button on the page that follows the login. After setting up GitHub authorization, future login screens will have that as an option: <***Log in with github-auth***> button.

5.1.2.5 Configure an External LDAP Directory

5.1.2.5.1 How to connect your cluster to an external LDAP directory

This guide shows you how to configure your DKP cluster so that users can log in with the credentials stored in an external LDAP directory service.

442 https://docs.github.com/en/organizations/organizing-members-into-teams/changing-team-visibility

5.1.2.5.2 Add LDAP connector

Each LDAP directory is set up in its own specific manner, so these steps are important. The LDAP authentication mechanism can be added using the CLI or the UI.

> The following example does not cover all possible configurations. Refer to the Dex LDAP connector reference documentation[443] for more details.

The example below configures a DKP cluster to connect to the Online LDAP Test Server[444] and for demonstration purposes, the configuration shown uses `insecureNoSSL: true`. In production, you should protect LDAP communication with a properly-configured transport layer security (TLS). When using TLS, the admin can add `insecureSkipVerify: true` to `spec.ldap` to skip server certificate verification, if needed.

Choose whether to establish an external LDAP globally, or for a specific workspace.

Global LDAP - identity provider serves all workspaces

Create and apply the following objects:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: ldap-password
  namespace: kommander
type: Opaque
stringData:
  password: password
---
apiVersion: dex.mesosphere.io/v1alpha1
kind: Connector
metadata:
  name: ldap
  namespace: kommander
spec:
  enabled: true
  type: ldap
  displayName: LDAP Test
  ldap:
    host: ldap.forumsys.com:389
    insecureNoSSL: true
    bindDN: cn=read-only-admin,dc=example,dc=com
```

443 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/ldap.md
444 https://www.forumsys.com/tutorials/integration-how-to/ldap/online-ldap-test-server/

```
    bindSecretRef:
      name: ldap-password
    userSearch:
      baseDN: dc=example,dc=com
      filter: "(objectClass=inetOrgPerson)"
      username: uid
      idAttr: uid
      emailAttr: mail
    groupSearch:
      baseDN: dc=example,dc=com
      filter: "(objectClass=groupOfUniqueNames)"
      userMatchers:
      - userAttr: DN
        groupAttr: uniqueMember
      nameAttr: ou
EOF
```

The value for the LDAP connector `spec:displayName` (here **LDAP Test**) appears on the login button for this identity provider in the DKP UI. Choose a name that is meaningful for users.

Workspace LDAP - identity provider serves a specific workspace

Establish LDAP for a specific workspace in the scope of Multi-Tenancy in DKP (see page 712).

Create and apply the following objects:

1. Obtain the workspace name for which you are establishing an LDAP authentication server:

   ```
   kubectl get workspaces
   ```

   Note down the value under the `WORKSPACE NAMESPACE` column.

2. Set the WORKSPACE_NAMESPACE environment variable to that namespace:

   ```
   export WORKSPACE_NAMESPACE=<your-namespace>
   ```

3. Create and apply the following objects on that workspace:

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: v1
   kind: Secret
   ```

```
metadata:
  name: ldap-password
  namespace: ${WORKSPACE_NAMESPACE}
type: Opaque
stringData:
  password: password
---
apiVersion: dex.mesosphere.io/v1alpha1
kind: Connector
metadata:
  name: ldap
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  enabled: true
  type: ldap
  displayName: LDAP Test
  ldap:
    host: ldap.forumsys.com:389
    insecureNoSSL: true
    bindDN: cn=read-only-admin,dc=example,dc=com
    bindSecretRef:
      name: ldap-password
    userSearch:
      baseDN: dc=example,dc=com
      filter: "(objectClass=inetOrgPerson)"
      username: uid
      idAttr: uid
      emailAttr: mail
    groupSearch:
      baseDN: dc=example,dc=com
      filter: "(objectClass=groupOfUniqueNames)"
      userMatchers:
      - userAttr: DN
        groupAttr: uniqueMember
      nameAttr: ou
EOF
```

The value for the LDAP connector `spec:displayName` (here **LDAP Test**) appears on the login button for this identity provider in the DKP UI. Choose a name that is meaningful for users.

5.1.2.5.2.1 Test the LDAP Connector

1. Retrieve a list of connectors:

```
kubectl get connector.dex.mesosphere.io -A
```

2. Run the following command to verify that the LDAP connector was created successfully:

```
kubectl get Connector.dex.mesosphere.io -n kommander <LDAP-CONNECTOR-NAME> -o
yaml
```

5.1.2.5.3 Log In

Global LDAP - identity provider serves all workspaces

1. Visit `https://<YOUR-CLUSTER-HOST>/token` and initiate a login flow.
2. On the login page, choose the `Log in with <ldap-name>` button.
3. Enter the LDAP credentials, and log in.

> UI - While LDAP authentication has been enabled, additional access rights will need to be configured through the Add Identity Provider screen in the UI using the documentation for Granting Access to Kubernetes and Kommander Resources (see page 596).

Workspace LDAP - identity provider serves a specific workspace

1. Generate a Dedicated URL Login for Each Tenant (see page 714).
2. On the login page, choose the `Log in with <ldap-name>` button.
3. Enter the LDAP credentials, and log in.

> UI - While LDAP authentication has been enabled, additional access rights will need to be configured through the Add Identity Provider screen in the UI using the documentation for Granting Access to Kubernetes and Kommander Resources (see page 596).

5.1.2.5.3.1 Next Step:

LDAP Troubleshooting (see page 618)

5.1.2.5.4 LDAP Troubleshooting

If the Dex LDAP connector configuration is not quite right from the start, debug the problem and iterate on it. The Dex log output contains helpful error messages, as indicated by the following examples.

5.1.2.5.4.1 Errors During Dex Startup

If the Dex configuration fragment provided results in an invalid Dex config, Dex does not properly start up. In that case, reviewing the Dex logs will provide error details. Use the following command to retrieve the Dex logs:

```
kubectl logs -f dex-66675fcb7c-snxb8  -n kommander
```

You may see an error similar to the following:

```
error parse config file /etc/dex/cfg/config.yaml: error unmarshaling JSON: parse
connector config: illegal base64 data at input byte 0
```

Another reason for Dex not starting up correctly is that `https://<YOUR-CLUSTER-HOST>/token` throws a 5xx HTTP error response after timing out.

5.1.2.5.4.2 Errors Upon Login

Most problems with the Dex LDAP connector configuration become apparent only after a login attempt. A login failing from misconfiguration will result in an error page showing only `Internal Server Error` and `Login error`. You can then usually find the root cause by reading the Dex log, as shown in the following example:

```
kubectl logs -f dex-5d55b6b94b-9pm2d -n kommander
```

You can look for output similar to this example:

```
[...]
time="2019-07-29T13:03:57Z" level=error msg="Failed to login user: failed to connect:
LDAP Result Code 200 \"Network Error\": dial tcp: lookup freeipa.example.com on
10.255.0.10:53: no such host"
```

Here, the directory's DNS name was misconfigured, which should be easy to address.

A more difficult problem occurs when a login through Dex through LDAP fails because Dex was not able to find the specified user unambiguously in the directory. That could be the result of an invalid LDAP user search configuration. Here's an example error message from the Dex log:

```
time="2019-07-29T14:21:27Z" level=info msg="performing ldap search
cn=users,cn=compat,dc=demo1,dc=freeipa,dc=org sub (&(objectClass=posixAccount)
(uid=employee))"
time="2019-07-29T14:21:27Z" level=error msg="Failed to login user: ldap: filter
returned multiple (2) results: \"(&(objectClass=posixAccount)(uid=employee))\""
```

Solving problems like this requires you to review the directory structure carefully. (Directory structures can be very different between different LDAP setups.) Then you must carefully assemble a user search configuration matching the directory structure.

Notably, with some directories, it can be hard to distinguish between the cases “properly configured, and user not found” (login fails in an expected way) and “not properly configured, and therefore user not found” (login fails in an unexpected way).

5.1.2.5.4.3 Example for Successful Login

For comparison, here are some sample log lines issued by Dex after successful login:

```
time="2019-07-29T15:35:51Z" level=info msg="performing ldap search
cn=accounts,dc=demo1,dc=freeipa,dc=org sub (&(objectClass=posixAccount)
(uid=employee))"
time="2019-07-29T15:35:52Z" level=info msg="username \"employee\" mapped to entry
uid=employee,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org"
time="2019-07-29T15:35:52Z" level=info msg="login successful: connector \"ldap\",
username=\"\", email=\"employee@demo1.freeipa.org\", groups=[]"
```

5.1.3 Kubectl API Access Using an Identity Provider

After installing DKP, a single user with admin rights and static credentials is available. However, static credentials are hard to manage and replace.

To allow other users and user groups to access your environment, D2iQ recommends setting up an external Identity Provider (see page 609). Users added through an identity provider do not have static credentials, but have to generate a token to gain access to your environment’s `kubectl` API. This token ensures that certificates are rotated continuously for security reasons.

You have **two options** for the generation of this token:

Method	How Often Does the User Have to Generate a Token?
Generate a token	User must log in with credentials. Then manually generate a `kubeconfig` file with a fresh token every 24 hours.

Method	How Often Does the User Have to Generate a Token?
Enable the Konvoy Async Plugin	User configures the `Konvoy Async Plugin`, so the authentication is routed through Dex's `oidc` and the token is generated automatically. By enabling the plugin, the user is routed to an additional login procedure for authentication, but they no longer have to generate a token manually in the UI.

The instructions for either generating a token manually or enabling the Konvoy Async Plugin differ slightly depending on whether you configured the Identity Provide globally for **All Workspaces**, or individually for a **Single Workspace** (see page 610). Here is an overview:

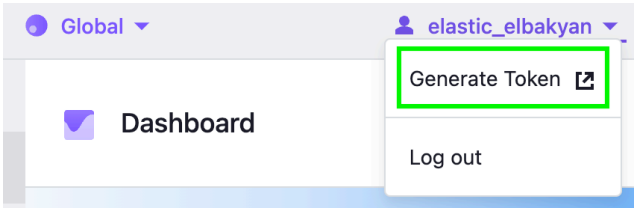
5.1.3.1 Configure Token Authentication for ***Global*** Identity Providers

In this scenario, the Identity Provider serves all workspaces.

Manually generate a token every 24 hours.

You must create a new token every 24 hours:

1. Log in to the DKP UI with your credentials.
2. Select your username.
3. Select **Generate Token**.

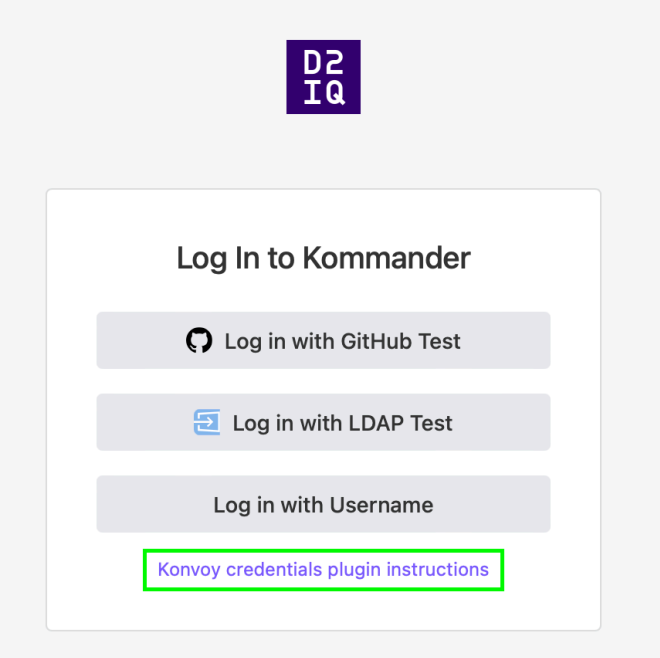


4. Log in again.
5. If there are several clusters, select the target cluster.
6. Follow the instructions on the displayed page.

Enable the Konvoy Async Plugin to automatically update the token.

> As a prerequisite, you or a global admin must have configured an Identity Provider to see this option.

1. Open the login URL.
2. You see several options to authenticate, select the **Konvoy credentials plugin instructions**:



3. Follow the instructions on the displayed “(Konvoy) Credentials plugin instructions” page.

If you use **Method 1** of the instructions documented in the “(Konvoy) Credentials plugin instructions”, you will download a `kubeconfig` file that includes the contexts for all clusters.

Alternatively, you can use **Method 2** to create a `kubeconfig` file *per cluster*, and use the `--kubeconfig=` flag or `export KUBECONFIG=` commands to switch between clusters.

> **Important Warning for Method 2:**
> The **Set profile name** field is not optional if you have *multiple clusters* in your environment. Ensure you change the name of the profile for each cluster for which you want to generate a `kubeconfig` file. Otherwise, all clusters will use the same token, which makes cluster authentication vulnerable and could let users access clusters for which they do not have authorization.

5.1.3.2 Configure Token Authentication for ***Workspace*** Identity Providers

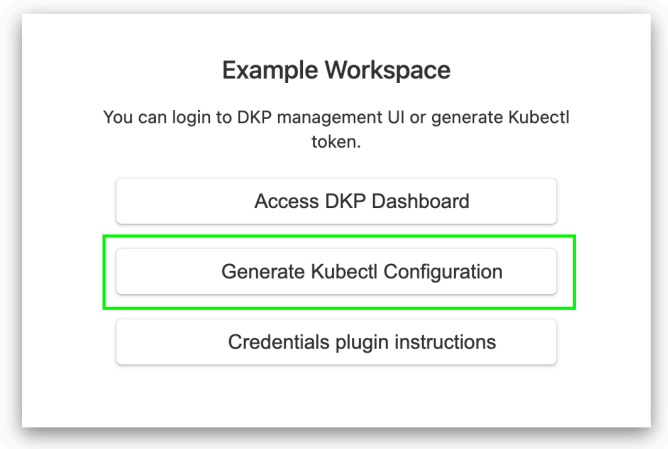
In this scenario, the Identity Provider serves a specific workspace or tenant, as explained in Multi-Tenancy in DKP (see page 712).

Configure Token Authentication

Manually generate a token every 24 hours.

You must create a new token every 24 hours:

1. Open the login link you obtained from the global administrator, which they generated for your workspace/tenant (see page 714).
2. Select **Generate Kubectl Configuration**:

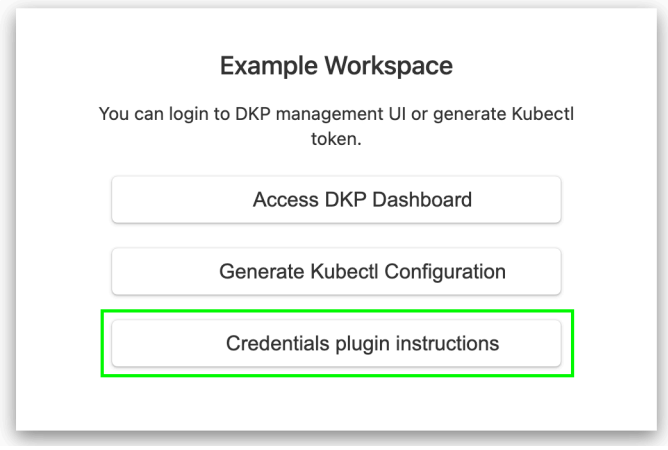


3. If there are several clusters in the workspace, select the cluster for which you want to generate a token.
4. Log in with your credentials.
5. Follow the instructions on the displayed page.

Enable the Konvoy Async Plugin to automatically update the token.

> As a prerequisite, you or a global admin must have configured a **workspace-scoped** Identity Provider to see this option.

1. Open the login link you obtained from the global administrator, which they generated for your workspace/tenant (see page 714).
2. Select **Credentials plugin instructions**:



3. Follow the instructions on the displayed “(Konvoy) Credentials plugin instructions” page.

If you use **Method 1** of the instructions documented in the "(Konvoy) Credentials plugin instructions", you will download a `kubeconfig` file that includes the contexts for all clusters.

Alternatively, you can use **Method 2** to create a `kubeconfig` file *per cluster*, and use the `--kubeconfig=` flag or `export KUBECONFIG=` commands to switch between clusters.

> **Important Warning for Method 2:**
> The **Set profile name** field is not optional if you have *multiple clusters* in your environment. Ensure you change the name of the profile for each cluster for which you want to generate a `kubeconfig` file. Otherwise, all clusters will use the same token, which makes cluster authentication vulnerable and could let users access clusters for which they do not have authorization.

5.1.3.3 FAQs

What happens if I have several clusters in my environment?

Each cluster has a unique API server address, and each cluster requires a unique token.

There are several things you can do to have access to several clusters.

If you decide to manually create a `kubeconfig` file with a fresh token every 24 hours, you must do this for each cluster in your environment.

If you decide to enable the Konvoy Async Plugin to automatically refresh the token, you can use the **Method 1** steps documented in the "(Konvoy) Credentials plugin instructions" page to download a `kubeconfig` that includes the contexts for all clusters.

Alternatively, you can use the **Method 2** steps documented in the "(Konvoy) Credentials plugin instructions" page to create a `kubeconfig` file *per cluster*, and use the `--kubeconfig=` flag or `export KUBECONFIG=` commands to switch between clusters.

> **Important Warning for Method 2:**
> The **Set profile name** field is not optional if you have *multiple clusters* in your environment. Ensure you change the name of the profile for each cluster for which you want to generate a `kubeconfig` file. Otherwise, all clusters will use the same token, which makes cluster authentication vulnerable and could let users access clusters for which they do not have authorization.

See https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/ for more information on switching between contexts or using the `--kubeconfig=` flag for multi-cluster environment management.

What happens if I have several workspaces, and I have dedicated Identity Providers for some or all of them?

DKP supports multi-tenancy (see page 712) by enabling administrators to configure dedicated Identity Providers per workspace/tenant. Given that a user employs a dedicated workspace login URL (see page 714) to access the DKP UI, the user only sees the login options (and IdPs) available in said workspace.

What is a kubeconfig file?

A file that is used to configure access to clusters is called a `kubeconfig` file.

The kubectl and DKP command-line tools use `kubeconfig` files to find the information it needs to choose a cluster and communicate with the API server of a cluster.

In DKP, a `kubeconfig` file is created automatically when you create a cluster, but it requires a valid token to obtain access to it.

See Provide Context for Commands with a kubeconfig File (see page 106) for more information.

5.1.4 Infrastructure Providers

Enterprise Gov Advanced

Infrastructure providers, like AWS, Azure, and vSphere, provide the infrastructure for your Management clusters. You may have many accounts for a single infrastructure provider. To automate cluster provisioning, DKP needs authentication keys for your preferred infrastructure provider.

To provision new clusters and manage them, DKP also needs infrastructure provider credentials. Currently, you can create infrastructure providers records for AWS (see page 0), Azure (see page 775), and vSphere (see page 0) in the DKP user interface.

> Infrastructure provider credentials are configured in each workspace. The name you assign must be unique across all namespaces in your cluster.

5.1.4.1 View and Modify Infrastructure Providers

You can use the DKP user interface to view, create, and delete infrastructure provider records.

1. From the top menu bar, select your target workspace.
2. Select **Infrastructure Providers** in the **Administration** section of the sidebar menu.

Refer to the information in the linked topics that follow for more information about creating infrastructure providers and clusters.

5.1.4.1.1 AWS

- Configure AWS Provider with Role Credentials (see page 625) (Recommended if using AWS)

- Configure AWS Provider with Static Credentials (see page 632)

5.1.4.1.2 Azure

- Create an Azure Infrastructure Provider in the DKP UI (see page 639)
- Create a Managed Azure Cluster from the DKP UI (see page 775)

5.1.4.1.3 vSphere

- Create a vSphere Infrastructure Provider in the DKP UI (see page 640)
- Create a Managed Cluster on vSphere from the DKP UI (see page 776)

5.1.4.1.4 VMware Cloud Director (VCD)

- Create a VMware Cloud Director Infrastructure Provider in the DKP UI (see page 640)
- Create a Managed Cluster on VCD Using the DKP UI (see page 780)

5.1.4.2 Delete an infrastructure provider

Before deleting an infrastructure provider, DKP verifies whether any existing managed clusters were created using this provider. You cannot delete an infrastructure provider until you delete all clusters created with that infrastructure provider. This ensures DKP has access to your infrastructure provider to remove all resources created for a managed cluster.

5.1.4.3 Configure an AWS Infrastructure Provider with a User Role

Enterprise Gov Advanced

Configure your provider to add resources to your AWS account

> We highly recommend using the role-based method as this is more secure.

> The role authentication method can only be used if your management cluster is running in AWS.

For more flexible credential configuration, we offer a role-based authentication[445] method with an optional External ID for third party access.

445 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html

5.1.4.3.1 Create a Role Manually

The role should grant permissions to create the following resources in the AWS account:

- EC2 Instances
- VPC
- Subnets
- Elastic Load Balancer (ELB)
- Internet Gateway
- NAT Gateway
- Elastic Block Storage (EBS) Volumes
- Security Groups
- Route Tables
- IAM Roles

The user you delegate from your role must have a minimum set of permissions. Below is the minimal IAM policy required:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AllocateAddress",
        "ec2:AssociateRouteTable",
        "ec2:AttachInternetGateway",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:CreateInternetGateway",
        "ec2:CreateNatGateway",
        "ec2:CreateRoute",
        "ec2:CreateRouteTable",
        "ec2:CreateSecurityGroup",
        "ec2:CreateSubnet",
        "ec2:CreateTags",
        "ec2:CreateVpc",
        "ec2:ModifyVpcAttribute",
        "ec2:DeleteInternetGateway",
        "ec2:DeleteNatGateway",
        "ec2:DeleteRouteTable",
        "ec2:DeleteSecurityGroup",
        "ec2:DeleteSubnet",
        "ec2:DeleteTags",
        "ec2:DeleteVpc",
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeAddresses",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeInstances",
        "ec2:DescribeInternetGateways",
```

```
        "ec2:DescribeImages",
        "ec2:DescribeNatGateways",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeNetworkInterfaceAttribute",
        "ec2:DescribeRouteTables",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeVolumes",
        "ec2:DetachInternetGateway",
        "ec2:DisassociateRouteTable",
        "ec2:DisassociateAddress",
        "ec2:ModifyInstanceAttribute",
        "ec2:ModifyNetworkInterfaceAttribute",
        "ec2:ModifySubnetAttribute",
        "ec2:ReleaseAddress",
        "ec2:RevokeSecurityGroupIngress",
        "ec2:RunInstances",
        "ec2:TerminateInstances",
        "tag:GetResources",
        "elasticloadbalancing:AddTags",
        "elasticloadbalancing:CreateLoadBalancer",
        "elasticloadbalancing:ConfigureHealthCheck",
        "elasticloadbalancing:DeleteLoadBalancer",
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeLoadBalancerAttributes",
        "elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",
        "elasticloadbalancing:DescribeTags",
        "elasticloadbalancing:ModifyLoadBalancerAttributes",
        "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
        "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
        "elasticloadbalancing:RemoveTags",
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeInstanceRefreshes",
        "ec2:CreateLaunchTemplate",
        "ec2:CreateLaunchTemplateVersion",
        "ec2:DescribeLaunchTemplates",
        "ec2:DescribeLaunchTemplateVersions",
        "ec2:DeleteLaunchTemplate",
        "ec2:DeleteLaunchTemplateVersions",
        "ec2:DescribeKeyPairs"
      ],
      "Resource": ["*"]
    },
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:CreateAutoScalingGroup",
        "autoscaling:UpdateAutoScalingGroup",
        "autoscaling:CreateOrUpdateTags",
        "autoscaling:StartInstanceRefresh",
```

```
        "autoscaling:DeleteAutoScalingGroup",
        "autoscaling:DeleteTags"
      ],
      "Resource": [
        "arn:*:autoscaling:*:*:autoScalingGroup:*:autoScalingGroupName/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/autoscaling.amazonaws.com/
AWSServiceRoleForAutoScaling"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "autoscaling.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/elasticloadbalancing.amazonaws.com/
AWSServiceRoleForElasticLoadBalancing"
      ],
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "elasticloadbalancing.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/spot.amazonaws.com/
AWSServiceRoleForEC2Spot"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "spot.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:PassRole"],
      "Resource": ["arn:*:iam::*:role/*.cluster-api-provider-aws.sigs.k8s.io"]
    },
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret",
        "secretsmanager:DeleteSecret",
```

```
        "secretsmanager:TagResource"
      ],
      "Resource": ["arn:*:secretsmanager:*:*:secret:aws.cluster.x-k8s.io/*"]
    },
    {
      "Effect": "Allow",
      "Action": ["ssm:GetParameter"],
      "Resource": ["arn:*:ssm:*:*:parameter/aws/service/eks/optimized-ami/*"]
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/eks.amazonaws.com/
AWSServiceRoleForAmazonEKS"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "eks.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/eks-nodegroup.amazonaws.com/
AWSServiceRoleForAmazonEKSNodegroup"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "eks-nodegroup.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:aws:iam::*:role/aws-service-role/eks-fargate-pods.amazonaws.com/
AWSServiceRoleForAmazonEKSForFargate"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "eks-fargate.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:GetRole", "iam:ListAttachedRolePolicies"],
      "Resource": ["arn:*:iam::*:role/*"]
    },
    {
      "Effect": "Allow",
      "Action": ["iam:GetPolicy"],
      "Resource": ["arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"]
    },
```

```
{
  "Effect": "Allow",
  "Action": [
    "eks:DescribeCluster",
    "eks:ListClusters",
    "eks:CreateCluster",
    "eks:TagResource",
    "eks:UpdateClusterVersion",
    "eks:DeleteCluster",
    "eks:UpdateClusterConfig",
    "eks:UntagResource",
    "eks:UpdateNodegroupVersion",
    "eks:DescribeNodegroup",
    "eks:DeleteNodegroup",
    "eks:UpdateNodegroupConfig",
    "eks:CreateNodegroup",
    "eks:AssociateEncryptionConfig"
  ],
  "Resource": ["arn:*:eks:*:*:cluster/*", "arn:*:eks:*:*:nodegroup/*/*/*"]
},
{
  "Effect": "Allow",
  "Action": [
    "eks:ListAddons",
    "eks:CreateAddon",
    "eks:DescribeAddonVersions",
    "eks:DescribeAddon",
    "eks:DeleteAddon",
    "eks:UpdateAddon",
    "eks:TagResource",
    "eks:DescribeFargateProfile",
    "eks:CreateFargateProfile",
    "eks:DeleteFargateProfile"
  ],
  "Resource": ["*"]
},
{
  "Effect": "Allow",
  "Action": ["iam:PassRole"],
  "Resource": ["*"],
  "Condition": {
    "StringEquals": { "iam:PassedToService": "eks.amazonaws.com" }
  }
},
{
  "Effect": "Allow",
  "Action": ["kms:CreateGrant", "kms:DescribeKey"],
  "Resource": ["*"],
  "Condition": {
    "ForAnyValue:StringLike": {
      "kms:ResourceAliases": "alias/cluster-api-provider-aws-*"
    }
```

```
      }
    }
  ]
}
```

Make sure to also add a correct trust relationship[446] to the created role. This example allows everyone within the same account to `AssumeRole` with the created role. `YOURACCOUNTRESTRICTION` must be replaced with the AWS Account ID you would like to `AssumeRole` from.

IMPORTANT: Never add a `*/` wildcard. This opens your account to the whole world.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com",
        "AWS": "arn:aws:iam::YOURACCOUNTRESTRICTION:root"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

To use the role created, attach the following policy to the role which is already attached to your Managed or Attached cluster. Replace `YOURACCOUNTRESTRICTION` with the AWS Account ID where the role you like to `AssumeRole` is saved. Also, replace `THEROLEYOUCREATED` with the AWS Role name.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AssumeRoleKommander",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::YOURACCOUNTRESTRICTION:role/THEROLEYOUCREATED"
    }
  ]
}
```

446 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html

5.1.4.3.2 Create an AWS Infrastructure Provider in DKP

1. From the top menu bar, select your target workspace.
2. Select **Infrastructure Providers** in the **Administration** section of the sidebar menu.
3. Select the **Add Infrastructure Provider** button.
4. Select the **Amazon Web Services (AWS)** option.
5. Ensure **Role** is selected as the **Authentication Method**.
6. Enter a name for your infrastructure provider. Select a name that matches the AWS user.
7. Enter the **Role ARN**.
8. You can add an **External ID** if you share the Role with a 3rd party. External IDs secure your environment from accidentally used roles. Read more about External IDs[447].
9. Select **Save** to save your provider.

5.1.4.4 Configure an AWS Infrastructure Provider with Static Credentials

Enterprise Gov Advanced

When configuring an infrastructure provider with static credentials, you need an access ID and secret key for a user with a set of minimum capabilities.

5.1.4.4.1 Create a New User Using CLI Commands

You will need to have the AWS CLI utility installed[448]. Create a new user with the AWS CLI commands below:

```
aws iam create-user --user-name Kommander
```

```
aws iam create-policy --policy-name kommander-policy --policy-document
'{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Action":
["ec2:AllocateAddress","ec2:AssociateRouteTable","ec2:AttachInternetGateway","ec2:AuthorizeSecurityG
roupIngress","ec2:CreateInternetGateway","ec2:CreateNatGateway","ec2:CreateRoute","ec2:CreateRouteTa
ble","ec2:CreateSecurityGroup","ec2:CreateSubnet","ec2:CreateTags","ec2:CreateVpc","ec2:ModifyVpcAtt
ribute","ec2:DeleteInternetGateway","ec2:DeleteNatGateway","ec2:DeleteRouteTable","ec2:DeleteSecurit
yGroup","ec2:DeleteSubnet","ec2:DeleteTags","ec2:DeleteVpc","ec2:DescribeAccountAttributes","ec2:Des
cribeAddresses","ec2:DescribeAvailabilityZones","ec2:DescribeInstances","ec2:DescribeInternetGateway
s","ec2:DescribeImages","ec2:DescribeNatGateways","ec2:DescribeNetworkInterfaces","ec2:DescribeNetwo
rkInterfaceAttribute","ec2:DescribeRouteTables","ec2:DescribeSecurityGroups","ec2:DescribeSubnets","
ec2:DescribeVpcs","ec2:DescribeVpcAttribute","ec2:DescribeVolumes","ec2:DetachInternetGateway","ec2:
DisassociateRouteTable","ec2:DisassociateAddress","ec2:ModifyInstanceAttribute","ec2:ModifyNetworkIn
terfaceAttribute","ec2:ModifySubnetAttribute","ec2:ReleaseAddress","ec2:RevokeSecurityGroupIngress",
"ec2:RunInstances","ec2:TerminateInstances","tag:GetResources","elasticloadbalancing:AddTags","elast
icloadbalancing:CreateLoadBalancer","elasticloadbalancing:ConfigureHealthCheck","elasticloadbalancin
g:DeleteLoadBalancer","elasticloadbalancing:DescribeLoadBalancers","elasticloadbalancing:DescribeLoa
dBalancerAttributes","elasticloadbalancing:ApplySecurityGroupsToLoadBalancer","elasticloadbalancing:
DescribeTags","elasticloadbalancing:ModifyLoadBalancerAttributes","elasticloadbalancing:RegisterInst
```

447 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-user_externalid.html
448 https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html

ancesWithLoadBalancer","elasticloadbalancing:DeregisterInstancesFromLoadBalancer","elasticloadbalancing:RemoveTags","autoscaling:DescribeAutoScalingGroups","autoscaling:DescribeInstanceRefreshes","ec2:CreateLaunchTemplate","ec2:CreateLaunchTemplateVersion","ec2:DescribeLaunchTemplates","ec2:DescribeLaunchTemplateVersions","ec2:DeleteLaunchTemplate","ec2:DeleteLaunchTemplateVersions","ec2:DescribeKeyPairs"],"Resource":["*"]},{"Effect":"Allow","Action":["autoscaling:CreateAutoScalingGroup","autoscaling:UpdateAutoScalingGroup","autoscaling:CreateOrUpdateTags","autoscaling:StartInstanceRefresh","autoscaling:DeleteAutoScalingGroup","autoscaling:DeleteTags"],"Resource":["arn:*:autoscaling:*:*:autoScalingGroup:*:autoScalingGroupName/*"]},{"Effect":"Allow","Action":["iam:CreateServiceLinkedRole"],"Resource":["arn:*:iam::*:role/aws-service-role/autoscaling.amazonaws.com/AWSServiceRoleForAutoScaling"],"Condition":{"StringLike":{"iam:AWSServiceName":"autoscaling.amazonaws.com"}}},{"Effect":"Allow","Action":["iam:CreateServiceLinkedRole"],"Resource":["arn:*:iam::*:role/aws-service-role/elasticloadbalancing.amazonaws.com/AWSServiceRoleForElasticLoadBalancing"],"Condition":{"StringLike":{"iam:AWSServiceName":"elasticloadbalancing.amazonaws.com"}}},{"Effect":"Allow","Action":["iam:CreateServiceLinkedRole"],"Resource":["arn:*:iam::*:role/aws-service-role/spot.amazonaws.com/AWSServiceRoleForEC2Spot"],"Condition":{"StringLike":{"iam:AWSServiceName":"spot.amazonaws.com"}}},{"Effect":"Allow","Action":["iam:PassRole"],"Resource":["arn:*:iam::*:role/*.cluster-api-provider-aws.sigs.k8s.io"]},{"Effect":"Allow","Action":["secretsmanager:CreateSecret","secretsmanager:DeleteSecret","secretsmanager:TagResource"],"Resource":["arn:*:secretsmanager:*:*:secret:aws.cluster.x-k8s.io/*"]},{"Effect":"Allow","Action":["ssm:GetParameter"],"Resource":["arn:*:ssm:*:*:parameter/aws/service/eks/optimized-ami/*"]},{"Effect":"Allow","Action":["iam:CreateServiceLinkedRole"],"Resource":["arn:*:iam::*:role/aws-service-role/eks.amazonaws.com/AWSServiceRoleForAmazonEKS"],"Condition":{"StringLike":{"iam:AWSServiceName":"eks.amazonaws.com"}}},{"Effect":"Allow","Action":["iam:CreateServiceLinkedRole"],"Resource":["arn:*:iam::*:role/aws-service-role/eks-nodegroup.amazonaws.com/AWSServiceRoleForAmazonEKSNodegroup"],"Condition":{"StringLike":{"iam:AWSServiceName":"eks-nodegroup.amazonaws.com"}}},{"Effect":"Allow","Action":["iam:CreateServiceLinkedRole"],"Resource":["arn:aws:iam::*:role/aws-service-role/eks-fargate-pods.amazonaws.com/AWSServiceRoleForAmazonEKSForFargate"],"Condition":{"StringLike":{"iam:AWSServiceName":"eks-fargate.amazonaws.com"}}},{"Effect":"Allow","Action":["iam:GetRole","iam:ListAttachedRolePolicies"],"Resource":["arn:*:iam::*:role/*"]},{"Effect":"Allow","Action":["iam:GetPolicy"],"Resource":["arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"]},{"Effect":"Allow","Action":["eks:DescribeCluster","eks:ListClusters","eks:CreateCluster","eks:TagResource","eks:UpdateClusterVersion","eks:DeleteCluster","eks:UpdateClusterConfig","eks:UntagResource","eks:UpdateNodegroupVersion","eks:DescribeNodegroup","eks:DeleteNodegroup","eks:UpdateNodegroupConfig","eks:CreateNodegroup","eks:AssociateEncryptionConfig"],"Resource":["arn:*:eks:*:*:cluster/*","arn:*:eks:*:*:nodegroup/*/*/*"]},{"Effect":"Allow","Action":["eks:ListAddons","eks:CreateAddon","eks:DescribeAddonVersions","eks:DescribeAddon","eks:DeleteAddon","eks:UpdateAddon","eks:TagResource","eks:DescribeFargateProfile","eks:CreateFargateProfile","eks:DeleteFargateProfile"],"Resource":["*"]},{"Effect":"Allow","Action":["iam:PassRole"],"Resource":["*"],"Condition":{"StringEquals":{"iam:PassedToService":"eks.amazonaws.com"}}},{"Effect":"Allow","Action":["kms:CreateGrant","kms:DescribeKey"],"Resource":["*"],"Condition":{"ForAnyValue:StringLike":{"kms:ResourceAliases":"alias/cluster-api-provider-aws-*"}}}]}'

```
aws iam attach-user-policy --user-name Kommander --policy-arn $(aws iam list-policies
--query 'Policies[?PolicyName==`kommander-policy`].Arn' | grep -o '".*"' | tr -d '"')
```

```
aws iam create-access-key --user-name Kommander
```

5.1.4.4.1.1 Using an Existing User

You can use an existing AWS user with credentials configured[449]. The user must be authorized to create the following resources in the AWS account:

449 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html

- EC2 Instances
- VPC
- Subnets
- Elastic Load Balancer (ELB)
- Internet Gateway
- NAT Gateway
- Elastic Block Storage (EBS) Volumes
- Security Groups
- Route Tables
- IAM Roles

Minimum IAM Policy Example

The following is the minimal IAM policy required:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AllocateAddress",
        "ec2:AssociateRouteTable",
        "ec2:AttachInternetGateway",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:CreateInternetGateway",
        "ec2:CreateNatGateway",
        "ec2:CreateRoute",
        "ec2:CreateRouteTable",
        "ec2:CreateSecurityGroup",
        "ec2:CreateSubnet",
        "ec2:CreateTags",
        "ec2:CreateVpc",
        "ec2:ModifyVpcAttribute",
        "ec2:DeleteInternetGateway",
        "ec2:DeleteNatGateway",
        "ec2:DeleteRouteTable",
        "ec2:DeleteSecurityGroup",
        "ec2:DeleteSubnet",
        "ec2:DeleteTags",
        "ec2:DeleteVpc",
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeAddresses",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeInstances",
        "ec2:DescribeInternetGateways",
        "ec2:DescribeImages",
        "ec2:DescribeNatGateways",
        "ec2:DescribeNetworkInterfaces",
```

```
      "ec2:DescribeNetworkInterfaceAttribute",
      "ec2:DescribeRouteTables",
      "ec2:DescribeSecurityGroups",
      "ec2:DescribeSubnets",
      "ec2:DescribeVpcs",
      "ec2:DescribeVpcAttribute",
      "ec2:DescribeVolumes",
      "ec2:DetachInternetGateway",
      "ec2:DisassociateRouteTable",
      "ec2:DisassociateAddress",
      "ec2:ModifyInstanceAttribute",
      "ec2:ModifyNetworkInterfaceAttribute",
      "ec2:ModifySubnetAttribute",
      "ec2:ReleaseAddress",
      "ec2:RevokeSecurityGroupIngress",
      "ec2:RunInstances",
      "ec2:TerminateInstances",
      "tag:GetResources",
      "elasticloadbalancing:AddTags",
      "elasticloadbalancing:CreateLoadBalancer",
      "elasticloadbalancing:ConfigureHealthCheck",
      "elasticloadbalancing:DeleteLoadBalancer",
      "elasticloadbalancing:DescribeLoadBalancers",
      "elasticloadbalancing:DescribeLoadBalancerAttributes",
      "elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",
      "elasticloadbalancing:DescribeTags",
      "elasticloadbalancing:ModifyLoadBalancerAttributes",
      "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
      "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
      "elasticloadbalancing:RemoveTags",
      "autoscaling:DescribeAutoScalingGroups",
      "autoscaling:DescribeInstanceRefreshes",
      "ec2:CreateLaunchTemplate",
      "ec2:CreateLaunchTemplateVersion",
      "ec2:DescribeLaunchTemplates",
      "ec2:DescribeLaunchTemplateVersions",
      "ec2:DeleteLaunchTemplate",
      "ec2:DeleteLaunchTemplateVersions",
      "ec2:DescribeKeyPairs"
    ],
    "Resource": ["*"]
  },
  {
    "Effect": "Allow",
    "Action": [
      "autoscaling:CreateAutoScalingGroup",
      "autoscaling:UpdateAutoScalingGroup",
      "autoscaling:CreateOrUpdateTags",
      "autoscaling:StartInstanceRefresh",
      "autoscaling:DeleteAutoScalingGroup",
      "autoscaling:DeleteTags"
    ],
```

```
      "Resource": [
        "arn:*:autoscaling:*:*:autoScalingGroup:*:autoScalingGroupName/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/autoscaling.amazonaws.com/
AWSServiceRoleForAutoScaling"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "autoscaling.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/elasticloadbalancing.amazonaws.com/
AWSServiceRoleForElasticLoadBalancing"
      ],
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "elasticloadbalancing.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/spot.amazonaws.com/
AWSServiceRoleForEC2Spot"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "spot.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:PassRole"],
      "Resource": ["arn:*:iam::*:role/*.cluster-api-provider-aws.sigs.k8s.io"]
    },
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:CreateSecret",
        "secretsmanager:DeleteSecret",
        "secretsmanager:TagResource"
      ],
      "Resource": ["arn:*:secretsmanager:*:*:secret:aws.cluster.x-k8s.io/*"]
```

```
    },
    {
      "Effect": "Allow",
      "Action": ["ssm:GetParameter"],
      "Resource": ["arn:*:ssm:*:*:parameter/aws/service/eks/optimized-ami/*"]
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/eks.amazonaws.com/
AWSServiceRoleForAmazonEKS"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "eks.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:*:iam::*:role/aws-service-role/eks-nodegroup.amazonaws.com/
AWSServiceRoleForAmazonEKSNodegroup"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "eks-nodegroup.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:CreateServiceLinkedRole"],
      "Resource": [
        "arn:aws:iam::*:role/aws-service-role/eks-fargate-pods.amazonaws.com/
AWSServiceRoleForAmazonEKSForFargate"
      ],
      "Condition": {
        "StringLike": { "iam:AWSServiceName": "eks-fargate.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["iam:GetRole", "iam:ListAttachedRolePolicies"],
      "Resource": ["arn:*:iam::*:role/*"]
    },
    {
      "Effect": "Allow",
      "Action": ["iam:GetPolicy"],
      "Resource": ["arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"]
    },
    {
      "Effect": "Allow",
      "Action": [
```

```
        "eks:DescribeCluster",
        "eks:ListClusters",
        "eks:CreateCluster",
        "eks:TagResource",
        "eks:UpdateClusterVersion",
        "eks:DeleteCluster",
        "eks:UpdateClusterConfig",
        "eks:UntagResource",
        "eks:UpdateNodegroupVersion",
        "eks:DescribeNodegroup",
        "eks:DeleteNodegroup",
        "eks:UpdateNodegroupConfig",
        "eks:CreateNodegroup",
        "eks:AssociateEncryptionConfig"
      ],
      "Resource": ["arn:*:eks:*:*:cluster/*", "arn:*:eks:*:*:nodegroup/*/*/*"]
    },
    {
      "Effect": "Allow",
      "Action": [
        "eks:ListAddons",
        "eks:CreateAddon",
        "eks:DescribeAddonVersions",
        "eks:DescribeAddon",
        "eks:DeleteAddon",
        "eks:UpdateAddon",
        "eks:TagResource",
        "eks:DescribeFargateProfile",
        "eks:CreateFargateProfile",
        "eks:DeleteFargateProfile"
      ],
      "Resource": ["*"]
    },
    {
      "Effect": "Allow",
      "Action": ["iam:PassRole"],
      "Resource": ["*"],
      "Condition": {
        "StringEquals": { "iam:PassedToService": "eks.amazonaws.com" }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["kms:CreateGrant", "kms:DescribeKey"],
      "Resource": ["*"],
      "Condition": {
        "ForAnyValue:StringLike": {
          "kms:ResourceAliases": "alias/cluster-api-provider-aws-*"
        }
      }
    }
  ]
```

```
}
```

5.1.4.4.1.2 Create an AWS Infrastructure Provider in DKP

Use the following steps to create an AWS infrastructure provider with static credentials as configured above:

1. In DKP, select the Workspace associated with the credentials you are adding.
2. Navigate to **Administration > Infrastructure Providers** and click the **Add Infrastructure Provider** button.
3. Select the Amazon Web Services (AWS) option.
4. Ensure **Static** is selected as the Authentication Method.
5. Enter a name for your infrastructure provider for later reference. Consider choosing a name that matches the AWS user.
6. Fill out the access ID and secret keys using the keys generated above.
7. Select **Save** to save your provider.

5.1.4.5 Create an Azure Infrastructure Provider in the DKP UI

Before you provision Azure clusters using the DKP UI, you must first create an Azure infrastructure provider to contain your Azure credentials:

1. Log in to the Azure command line:

   ```
   az login
   ```

2. Create an Azure Service Principal (SP) by running the following command:

   ```
   az ad sp create-for-rbac --role contributor --name "$(whoami)-konvoy" --
   scopes=/subscriptions/$(az account show --query id -o tsv)
   ```

3. Select **Infrastructure Providers** from the Dashboard menu.
4. Select **Add Infrastructure Provider**.
5. Workspaces (see page 678) If you are already in a workspace, the provider is automatically created in that workspace.
6. Select **Microsoft Azure**.
7. Add a **Name** for your Infrastructure Provider.
8. Copy and paste the following values into the indicated fields:
   - Copy the `id` output from the login command above and paste it into the **Subscription ID** field.
   - Copy the `tenant` used in Step 2 and paste it into the **Tenant ID** field.

- Copy the `appId` used in Step 2 and paste it into the **Client ID** field.
- Copy the `password` used in Step 2 and paste it into the **Client Secret** field.

9. Select **Save**.

5.1.4.6 Create a vSphere Infrastructure Provider in the DKP UI

Next, you must create a vSphere infrastructure provider:

1. Log in to your DKP Enterprise user interface and access the DKP home page.
2. Select **Infrastructure Providers** from the navigation menu on the left.
3. Select **Add Infrastructure Provider**.
4. Choose a workspace. (see page 678) If you are already in a workspace, the new infrastructure provider is automatically created in that workspace.
5. Select **vSphere** and add the following information:
   - Add a **Name** for your Infrastructure Provider.
   - Enter a valid vSphere vCenter user's name in the **Username** field.
   - Enter a valid vSphere vCenter user's password in the **Password** field.
   - Enter the vCenter Server URL in the **Host URL** field.
     This field should contain only the domain for the URL, such as `vcenter.ca1.your-org-platform.domain.cloud`. Do not specify the protocols `http://` or `https://` to avoid errors during cluster creation.
   - Enter a valid **TLS Certificate Thumbprint** value in the field.
     The TLS Certificate Thumbprint helps to create a secure connection to VMware vCenter. If you do not have a thumprint, your connection will be marked as insecure. This field is optional because you might not have a self-signed vCenter instance, and you only need the thumbprint if you do. The command to obtain this SHA-1 thumbprint for the vSphere's server's TLS certificate is listed under the field in the interface.
6. Select the **Save** button in the upper right corner of the page.

5.1.4.7 Create a VMware Cloud Director Infrastructure Provider in the DKP UI

Before you provision VMware Cloud Director (VCD) clusters using the DKP UI, you must:

- Fulfill the prerequisites (see page 1445) for VMware Cloud Director
- Create a VCD infrastructure provider to contain your credentials

5.1.4.7.1 Create a VCD Infrastructure Provider

1. Log in to your DKP Enterprise user interface.
2. Select **Infrastructure Providers** from the navigation menu on the left.

3. Select **Add Infrastructure Provider**.
4. Add a **Provider Name** for referencing this infrastructure provider.
5. Specify a **Refresh Token** name that you created in VCD.
   You can generate API Tokens[450] to grant programmatic access to VCD for both providers and tenant users. Automation scripts or third-party solutions use the API token to make requests of VCD on the user's behalf. You must generate, and name, the token within VCD, and also grant tenants the right to use and manage them.
6. Specify a **Site URL** value, which must begin with `https://`, for example, `"https://vcd.example.com"`. Do not use a trailing forward slash character.

VCD API token documentation[451] warns that you need to make a secure note of the Refresh Token, as it displays only one time, and cannot be retrieved afterward.

Editing a VCD infrastructure provider means that you are changing the credentials under which DKP connects to VMware Cloud Director. This can have negative effects on any existing clusters that use that infrastructure provider record.

To prevent errors, DKP first checks to see if there are any existing clusters for the selected infrastructure provider. If a VCD infrastructure provider has existing clusters, DKP displays an error message and *prevents* you from editing the infrastructure provider.

5.1.5 Add a Header, Footer, and Logo to Your DKP Implementation

Use the customizable Banners page to add header banners, footer banners, and select the colors for them. You can define header and/or footer banners for your DKP pages and turn them on and off, as needed.

DKP displays your header and footer banner in a default typeface and size, which cannot be changed.

450 https://blogs.vmware.com/cloudprovider/2022/03/cloud-director-api-token.html
451 https://blogs.vmware.com/cloudprovider/2022/03/cloud-director-api-token.html

5.1.5.1 Creating a Header Banner

The text you type in the **Text** field appears centered at the top of the screen. The text length is limited to 100 characters, including spaces. The text color is determined by the background color and automatically calculates an appropriate light or dark color for you.

The **Color** selection control uses the style of your browser for its color picker tool. This control allows you to select a color for your header banner in one of three ways:

- Enter the color's Hex code.
- Select a general color range, and then select a specific shade or tint. The color input uses the style of your browser for its color selection tool.
- Select the eyedropper, move it to a sample of the color you want and select once to select that color's location.

5.1.5.2 Creating a Footer Banner

The text you type in the **Text** field appears centered at the bottom of the screen.

The **Color** selection control uses the style of your browser for its color picker tool. This control allows you to select a color for your footer banner in one of three ways:

- Enter the color's Hex code.
- Select a general color range from the slider bar, and then select a specific shade or tint with your mouse cursor.
- Select the eyedropper, move it to a sample of the color you want and select once to select that color's location.

5.1.5.3 Adding Your Organization's Logo to the Header

Enterprise Gov Advanced

When you license and install DKP Enterprise or Gov Advanced, you also have the option to add your organization's logo to the header. The width of the header banner automatically adjusts to contain your logo. DKP automatically places your logo on the left side of the header, and centers it vertically.

Your logo graphic must meet the following criteria:

- Use a suggested file format - PNG, SVG, or JPEG.
- The file size cannot exceed 200 KB.

Error messages affecting the file to upload appear below the image in red, inside the shaded logo area.

To provide security against certain kinds of malicious activity, your browser has a “same-origin policy” for accessing resources. When you upload a file, the browser creates a unique identifier for the file. This will prevent you from selecting a file more than once.

5.1.5.3.1 Upload a logo image using drag-and-drop

1. Locate the desired file in the MacOS Finder or Windows File Explorer.
2. Drag and drop an image of the appropriate file type into the shaded area to see a preview of the image and display the filename.
   You can select the **X** button (upper-right) or the **Remove** link (lower-right) to clear the image, if needed.
3. Select the **Save** button in the upper-right corner to retain your changes.

You will not be able to select a file for drag-and-drop if it does not have a valid image format.

5.1.5.3.2 Upload a logo image by browsing your files

1. Select the Browse Files button.
2. Navigate to, and select, the desired file to see a preview of the image and display the filename.
   You can select the **X** button (upper-right) or the **Remove** link (lower-right) to clear the image, if needed.
3. Select the **Save** button in the upper-right corner to retain your changes.

5.2 Applications

This section includes information on the applications you can deploy in DKP.

5.2.1 AppDeployment Resources

Use AppDeployments to deploy, and customize platform, DKP catalog, and custom applications in your environment

An `AppDeployment` is a Custom Resource[452] created by DKP with the purpose of deploying applications (platform, DKP catalog and custom applications) in the management cluster, managed clusters, or both. Customers of both Essential and Enterprise products use `AppDeployments`, regardless of their setup (air-gapped, non-air-gapped, etc.), and their infrastructure provider.

When installing DKP, an `AppDeployment` resource is created for each enabled **Platform Application**. This `AppDeployment` resource references a `ClusterApp`, which then references the repository that contains a concrete declarative and preconfigured setup of an application, usually in the form of a `HelmRelease`. `ClusterApps` are cluster-scoped so that these platform applications are deployable to all workspaces or projects.

In the case of **DKP catalog** and **custom applications**, the `AppDeployment` references an `App` instead of a `ClusterApp`, which also references the repository containing the installation and deployment information. `Apps` are namespace-scoped and are meant to only be deployable to the workspace or project in which they have been created.

For example, this is the default `AppDeployment` for the Kube Prometheus Stack platform application:

```
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: kube-prometheus-stack-46.8.0
    kind: ClusterApp
```

5.2.1.1 Customization

5.2.1.1.1 Prerequisites

Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace where the cluster is attached:

```
export WORKSPACE_NAMESPACE=<your_workspace_namespace>
```

452 https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/

You are now able to copy the following commands without having to replace the placeholder with your workspace namespace every time you run a command.

5.2.1.1.2 Customize Your Application

If you want to customize an application, or change how a specific app is deployed, you can create a `ConfigMap` to change or add values to the information that is stored in the `HelmRelease`. Override the default configuration of an application by setting the `configOverrides` field on the `AppDeployment` to that `ConfigMap`. This overrides the configuration of the app for all clusters within the workspace.

For workspace applications, you can also enable and customize them on a per-cluster basis. Refer to the cluster-scoped configuration (see page 682) page for instructions on how to enable and customize an application per cluster in a given workspace.

This is an example, of how to customize the `AppDeployment` of Kube Prometheus Stack:

1. Provide the name of a `ConfigMap` with the custom configuration in the `AppDeployment`:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: kube-prometheus-stack-46.8.0
    kind: ClusterApp
  configOverrides:
    name: kube-prometheus-stack-overrides-attached
EOF
```

2. Create the `ConfigMap` with the name provided in the previous step, which provides the custom configuration on top of the default configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: kube-prometheus-stack-overrides-attached
data:
  values.yaml: |
    prometheus:
      prometheusSpec:
        storageSpec:
          volumeClaimTemplate:
```

```
            spec:
              resources:
                requests:
                  storage: 150Gi
EOF
```

5.2.1.1.3 Print and Review the Current State of an `AppDeployment` Resource

If you want to know how the `AppDeployment` resource is currently configured, use the commands below to print a table of the declared information. If the `AppDeployment` is configured for several clusters in a workspace, a column will display a list of the clusters.

5.2.1.1.3.1 Review all `AppDeployments` in a workspace

To review the state of the `AppDeployment` resource for a specific workspace, run the `get` command with the name of your workspace, as in this example:

```
dkp get appdeployments -w kommander-workspace
```

The output should contain a list of all your applications, here is an example:

```
NAME APP CLUSTERS
[...]
kube-oidc-proxy                kube-oidc-proxy-0.3.2           host-cluster
kube-prometheus-stack          kube-prometheus-stack-46.8.0    host-cluster
kubecost                       kubecost-0.35.1                 host-cluster
[...]
```

5.2.1.1.3.2 Review a specific `AppDeployment` of an application in a workspace

To review the state of a specific `AppDeployment` of an application, run the `get` command with the name of the application and your workspace, as in this example:

```
dkp get appdeployment kube-prometheus-stack -w kommander-workspace
```

The output should look similar to this:

```
NAME APP CLUSTERS
kube-prometheus-stack kube-prometheus-stack-46.8.0 host-cluster
```

5.2.1.1.4 Deployment Scope

In a single-cluster environment with an **Essential license**, `AppDeployments` enable customizing any platform application.

In a multi-cluster environment with an **Enterprise license**, `AppDeployments` enable workspace-level (see page 124), project-level (see page 720), and per-cluster deployment and customization of workspace applications (see page 682).

5.2.1.1.5 More Information

Refer to the CLI documentation (see page 1835) for more information on how to `create`, or `get` an `AppDeployment`.

5.2.2 Logging Stack Application Sizing Recommendations

# of worker nodes	Log Generating Load	Application	Suggested Configuration

50	1.4 MB/s	Logging Operator Logging Override Config	values.yaml: |-
			  clusterOutputs:
			  - name: loki
			  spec:
			  loki:
			  # change ${WORKSPACE_NAMESPACE} to the actual value of your workspace namespace
			  url: http://grafana-loki-loki-distributed-gateway.${WORKSPACE_NAMESPACE}.svc.cluster.local:80
			
			extract_kubernetes_labels: true
			
			configure_kubernetes_labels: true
			  buffer:
			  disabled: true
			
			retry_forever: false
			
			retry_max_times: 5
			  flush_mode: interval
			
			flush_interval: 10s
			
			flush_thread_count: 8
			  extra_labels:
			  log_source: kubernetes_container
			  fluentbit:
			  inputTail:
			  Mem_Buf_Limit: 512MB
			
			  fluentd:
			  bufferStorageVolume:
			  emptyDir:
			  medium: Memory
			  disablePvc: true
			  scaling:
			  replicas: 10
			  resources:

			<pre>requests:
			  memory: 1000Mi
			  cpu: 1000m
			limits:
			  memory: 2000Mi
			  cpu: 1000m</pre>

		Loki	<pre>ingester:
			  replicas: 10
			distributor:
			  replicas: 2</pre>

100	8.5 MB/s	Logging Operator Logging Override Config	

```
values.yaml: |-
  clusterOutputs:
    - name: loki
      spec:
        loki:
          # change ${WORKSPACE_NAMESPACE} to the actual value of your workspace namespace
          url: http://grafana-loki-loki-distributed-gateway.${WORKSPACE_NAMESPACE}.svc.cluster.local:80

          extract_kubernetes_labels: true

          configure_kubernetes_labels: true
          buffer:
            disabled: true

          retry_forever: false

          retry_max_times: 5
            flush_mode: interval

          flush_interval: 10s

          flush_thread_count: 8
          extra_labels:
            log_source: kubernetes_container
  fluentbit:
    inputTail:
      Mem_Buf_Limit: 512MB

  fluentd:
    bufferStorageVolume:
      emptyDir:
        medium: Memory
    disablePvc: true
    scaling:
      replicas: 15
    resources:
```

```
      requests:
        memory: 1000Mi
        cpu: 1000m
      limits:
        memory: 2000Mi
        cpu: 1000m
```

Loki Override Config

```
ingester:
  replicas: 12
distributor:
  replicas: 3
loki:
  structuredConfig:
    limits_config:
      ingestion_rate_mb:
1024

ingestion_burst_size_mb:
1024

per_stream_rate_limit:
30MB

per_stream_rate_limit_bu
rst: 60MB

max_streams_per_user: 0

max_global_streams_per_u
ser: 0

max_label_names_per_seri
es: 40
gateway:
  nginxConfig:
    httpSnippet: |-

client_max_body_size
50M;
    serverSnippet: |-

client_max_body_size
50M;
```

Refer to AppDeployment resources (see page 644) for information on how you customize your AppDeployments.

When configuring storage for `logging-operator-logging-overrides`, ensure that you create a ConfigMap in your workspace namespace for every cluster in that workspace.
Keep in mind that `logging-operator-logging-overrides` can only be configured via the CLI.

5.2.3 Rook Ceph Cluster Sizing Recommendations

# of worker nodes	Application	Suggested Configuration

50	Rook Ceph Cluster	

```
cephClusterSpec:
  labels:
    monitoring:

prometheus.kommander.d2iq.io/
select: "true"
  storage:
    storageClassDeviceSets:
      - name: rook-ceph-osd-
set1
        count: 4
        portable: true
        encrypted: false
        placement:

topologySpreadConstraints:
          - maxSkew: 1
            topologyKey:
topology.kubernetes.io/zone #
The nodes in the same rack have
the same
topology.kubernetes.io/zone
label.
            whenUnsatisfiable:
ScheduleAnyway
            labelSelector:
              matchExpressions:
                - key: app
                  operator: In
                  values:
                    - rook-
ceph-osd
                    - rook-
ceph-osd-prepare
          - maxSkew: 1
            topologyKey:
kubernetes.io/hostname
            whenUnsatisfiable:
ScheduleAnyway
            labelSelector:
              matchExpressions:
                - key: app
                  operator: In
                  values:
                    - rook-
ceph-osd
                    - rook-
ceph-osd-prepare
        volumeClaimTemplates:
```

```
          # If there are some
faster devices and some slower
devices, it is more efficient
to use
          # separate metadata,
wal, and data devices.
          # Refer https://
rook.io/docs/rook/v1.10/CRDs/
Cluster/pvc-cluster/#dedicated-
metadata-and-wal-device-for-
osd-on-pvc
          - metadata:
              name: data
            spec:
              resources:
                requests:
                  storage:
120Gi
              volumeMode: Block
              accessModes:
                - ReadWriteOnce
```

100

```
dkp:
  grafana-loki:
    additionalConfig:
      maxSize: "1000G"

cephClusterSpec:
  labels:
    monitoring:

prometheus.kommander.d2iq.io/
select: "true"
  storage:
    storageClassDeviceSets:
      - name: rook-ceph-osd-
set1
        count: 8
        portable: true
        encrypted: false
        placement:

topologySpreadConstraints:
          - maxSkew: 1
            topologyKey:
topology.kubernetes.io/zone #
The nodes in the same rack have
the same
topology.kubernetes.io/zone
label.
            whenUnsatisfiable:
ScheduleAnyway
            labelSelector:
              matchExpressions:
                - key: app
                  operator: In
                  values:
                    - rook-
ceph-osd
                    - rook-
ceph-osd-prepare
          - maxSkew: 1
            topologyKey:
kubernetes.io/hostname
            whenUnsatisfiable:
ScheduleAnyway
            labelSelector:
              matchExpressions:
                - key: app
                  operator: In
                  values:
```

```
                    - rook-
ceph-osd
                    - rook-
ceph-osd-prepare
        volumeClaimTemplates:
            - metadata:
              name: data
              spec:
                resources:
                  requests:
                    storage:
200Gi
                volumeMode:
Block
                accessModes:
                  -
ReadWriteOnce
```

Refer to AppDeployment resources (see page 644) for information on how you customize your AppDeployments.

To add more storage to `rook-ceph-cluster`, copy and paste the `storageClassDeviceSets` list from the `rook-ceph-cluster-1.10.3-d2iq-defaults` ConfigMap into your workspace where `rook-ceph-cluster` is present and then modify `count` and `volumeClaimTemplates.spec.resource.requests.storage`.

5.2.4 Manage an Application using the UI

Choose your license type for instructions on how to enable, and customize an application and then verify it has been deployed correctly.

- Disable an Application using the UI - Essential (see page 662)

5.2.4.1 Enterprise - Manage an Application using the UI

Enterprise Gov Advanced

> To use the CLI to deploy or uninstall applications, see Application Deployment (see page 667).

- Enable an Application using the UI (see page 657)
- Customize an Application using the UI (see page 658)
- Verify an Application using the UI (see page 659)
- Disable an Application using the UI (see page 660)

5.2.4.1.1 Enable an Application using the UI

Enterprise Gov Advanced

Follow these steps to enable your platform applications from the UI:

1. From the top menu bar, select your target workspace.
2. Select **Applications** from the sidebar to browse through the available applications from your configured repositories.
3. Select the three-dot button of the desired application card > **Enable**.
4. If available, select a version from the drop-down menu. This drop-down menu is only visible if there is more than one version to choose from.
5. Select the clusters where you want to deploy the application.
6. **For customizations only:** to override the default configuration values, select **Configuration** in the sidebar.
   Note: If there are customization *Overrides* at the workspace and cluster level, they are combined for implementation. *Cluster-level Overrides* take precedence over *Workspace Overrides*.
   a. To customize an application for all clusters in a workspace, copy your customized values into the text editor under **Workspace Application Configuration** or upload your YAML file that contains the values:

      ```
      someField: someValue
      ```

   b. To add a customization per cluster, copy the customized values into the text editor of each cluster under **Cluster Application Configuration Override** or upload your YAML file that contains the values:

```
someField: someValue
```

7. Confirm the details are correct, and then select the **Enable** button.

There may be dependencies between the applications, which are listed in the Workspace Platform Application Dependencies (see page 669) documentation. Review them carefully prior to customizing to ensure that the applications are deployed successfully.

5.2.4.1.1.1 Next topic:

Customize an App using the UI (see page 658)

5.2.4.1.2 Customize an Application using the UI

If you want to enable an application for the first time and customize it, refer to Enable an Application using the UI (see page 657).

You can customize the applications that are deployed to a workspace's cluster using the UI:

1. From the top menu bar, select your target workspace.
2. Select **Applications** from the sidebar menu to browse all applications.
3. In the **Application** card you want to customize, select the three dot menu and **Edit**.
4. To override the default configuration values, select **Configuration** in the sidebar.
   Note: If there are customization *Overrides* at the workspace and cluster level, they are combined for implementation. *Cluster-level Overrides* take precedence over *Workspace Overrides*.
   a. To customize an application for all clusters in a workspace, copy your customized values into the text editor under **Workspace Application Configuration** or upload your YAML file that contains the values:

      ```
      someField: someValue
      ```

   b. To add a customization per cluster, copy the customized values into the text editor of each cluster under **Cluster Application Configuration Override** or upload your YAML file that contains the values:

```
someField: someValue
```

5. Confirm the details are correct, and select **Save**.

5.2.4.1.2.1 Customize an App for a Specific Cluster from the **Clusters** page

You can also customize an application *for a specific cluster* from the **Clusters** view:

1. Select **Clusters** from the sidebar menu.
2. Selecting the target cluster.
3. Select the **Applications** tab.
4. Navigate to the target **Application** card.
5. Select the three-dot menu > **Edit**.

5.2.4.1.2.2 Next topic:

Verify an App using the UI (see page 659)

5.2.4.1.3 Verify an Application using the UI

Enterprise Gov Advanced

The application has now been enabled. Follow these steps to ensure the application was deployed correctly:

1. Select your target workspace from the top menu bar.
2. Select the cluster you want to verify from the left sidebar menu:
   a. Select **Management Cluster** if your target cluster is the Management Cluster Workspace.
   b. Otherwise, select **Clusters**, and choose your target cluster.
3. Select the **Applications** tab and navigate to the application you want to verify.
4. If the application was deployed successfully, the status **Deployed** appears in the application card. Otherwise, hover over the failed status to obtain more information on why the application failed to deploy.

It can take several minutes for the application to deploy completely. If the **Deployed** or **Failed** status is not displayed, the deployment process is not finished.

5.2.4.1.4 Disable an Application using the UI

Enterprise Gov Advanced

Follow these steps to disable an application with the UI:

1. From the top menu bar, select your target workspace.
2. Select **Applications** from the sidebar to browse through the available applications from your configured repositories.
3. Select the three-dot button of the desired application card > **Uninstall**.
4. Follow the instruction on the confirmation pop-up message, and select **Uninstall Application**.

5.2.4.2 Essential - Manage an Application using the UI

To use the CLI to deploy or uninstall applications, see Application Deployment (see page 667).

5.2.4.2.1 Enable an Application using the UI - Essential

Follow these steps to enable your platform applications from the UI in Kommander:

1. Select **Applications** from the sidebar to browse through the available applications from your configured repositories.
2. Select the three-dot button of the desired application card > **Enable**.
3. If available, select a version from the drop-down menu. This drop-down menu is only visible if there is more than one version to choose from.
4. **For customizations only:**
   - To override the default configuration values, select **Configuration** in the sidebar.
   - Copy your customized values into the text editor under **Workspace Application Configuration** or upload your YAML file that contains the values:

     ```
     someField: someValue
     ```
5. Confirm the details are correct, and then select the **Enable** button.

There may be dependencies between the applications, which are listed in the Workspace Platform Application Dependencies (see page 669) documentation. Review them carefully prior to customizing to ensure that the applications are deployed successfully.

5.2.4.2.1.1 Next topic:

Customize an App using the UI - Essential (see page 661)

5.2.4.2.2 Customize an Application using the UI - Essential

If you want to enable an application for the first time and customize it, refer to Enable an Application using the UI - Essential (see page 660).

You can customize the applications that are deployed to your Management Cluster using the UI:

1. Select **Applications** from the sidebar to browse all applications.
2. In the **Application** card you want to customize, select the three dot menu and **Edit**.
3. To override the default configuration values, select **Configuration** in the sidebar.
    a. To customize an application, copy your customized values into the text editor under **Workspace Application Configuration** or upload your YAML file that contains the values:

    ```
    someField: someValue
    ```

4. Confirm the details are correct, and select **Save**.

5.2.4.2.2.1 Next topic:

Verify an App via UI - Essential (see page 661)

5.2.4.2.3 Verify an Application via UI - Essential

The application has now been enabled. Follow these steps to ensure the application was deployed correctly:

1. Select **Management Cluster** from the sidebar.
2. Select the **Applications** tab and navigate to the application you want to verify.

3. If the application was deployed successfully, the status **Deployed** appears in the application card. Otherwise, hover over the failed status to obtain more information on why the application failed to deploy.

> It can take several minutes for the application to deploy completely. If the **Deployed** or **Failed** status is not displayed, the deployment process is not finished.

5.2.4.2.4 Disable an Application using the UI - Essential

Follow these steps to disable an application with the UI:

1. Select **Applications** from the sidebar to browse through the available applications from your configured repositories.
2. Select the three-dot button of the desired application card > **Uninstall**.
3. Follow the instruction on the confirmation pop-up message, and select **Uninstall Application**.

5.2.5 Platform Applications

How platform applications work

When attaching a cluster, DKP deploys certain platform applications on the newly attached cluster. Operators can use the DKP UI to customize which platform applications to deploy to the attached clusters in a given workspace. Refer to the Release Notes for the default DKP Applications (see page 1950) and their current versions.

5.2.5.1 Default Foundational Applications

These applications provide the foundation for all Platform Application capabilities and deployments on Managed Clusters. These applications must be enabled for any Platform Applications to work properly. For current DKP release Helm Values and DKP Values, refer to the Release Notes: DKP 2.7.0 Components and Applications (see page 1950) and related topics at the bottom of the page.

The foundational applications are comprised of the following Platform Applications:

- cert-manager[453]: Automates TLS certificate management and issuance.
- reloader[454]: A controller that watches changes on ConfigMaps and Secrets, and automatically triggers updates on the dependent applications.
- traefik[455]: Provides an HTTP reverse proxy and load balancer. Requires cert-manager and reloader.
- gitea[456]: Similar to GitHub and is installed using Helmchart

453 https://cert-manager.io/docs
454 https://github.com/stakater/Reloader
455 https://traefik.io/
456 https://docs.gitea.com/

- chartmuseum[457]: An Open source Helm Chart (collection of files that describe a set of Kubernetes resources) repository.
    - **Air-gapped environments only** - ChartMuseum is used on air-gapped installations to store the Helm Charts for Air-gapped installations. In non-air-gapped installations, the charts are fetched from upstream repositories and Chartmuseum is not installed.

Common Platform Application Name	APP ID
Cert-Manager	cert-manager
Logging Operator	logging-operator
Reloader	reloader
Traefik	traefik
Traefik ForwardAuth	traefik-forward-auth
ChartMuseum	chartmuseum[458]
Gitea	gitea

To see which applications are Enabled/Disabled in each category, verify the status:

```
kubectl get apps,clusterapps,appdeployments -A
```

After deployment (see page 667), applications will be enabled. To check whether enabled or not, connect to the attached cluster and watch the `HelmReleases` to verify the deployment. In this example, we are checking if `istio` got deployed correctly:

```
kubectl get helmreleases istio -n ${WORKSPACE_NAMESPACE} -w
```

You should eventually see the `HelmRelease` marked as `Ready`:

```
NAMESPACE              NAME        READY   STATUS                             AGE
workspace-test-vjsfq   istio       True    Release reconciliation succeeded   7m3s
```

457 https://chartmuseum.com/
458 https://github.com/helm/chartmuseum

5.2.5.2 Logging

Collects logs over time from Kubernetes and applications deployed on managed clusters. Also provides the ability to visualize and query the aggregated logs.

- fluent-bit[459]: Open source and multi-platform log processor tool which aims to be a generic Swiss knife for logs processing and distribution.
- grafana-logging[460]: Logging dashboard used to view logs aggregated to Grafana Loki.
- grafana-loki[461]: A horizontally-scalable, highly-available, multi-tenant log aggregation system inspired by Prometheus.
- logging-operator[462]: Automates the deployment and configuration of a Kubernetes logging pipeline.
- rook-ceph[463] and rook-ceph-cluster[464]: A Kubernetes-native high performance object store with an S3-compatible API that supports deploying into private and public cloud infrastructures.

Currently, the monitoring stack is deployed by default. The logging stack is not.

Common Platform Application Name	APP ID
Fluent Bit	fluent-bit
Grafana Logging	grafana-logging
Logging Operator	logging-operator
Grafana Loki (project)	project-grafana-loki
Rook Ceph	rook-ceph
Rook Ceph Cluster	rook-ceph-cluster

459 https://docs.fluentbit.io/manual/
460 https://grafana.com/oss/grafana/
461 https://grafana.com/oss/loki/
462 https://banzaicloud.com/docs/one-eye/logging-operator/
463 https://rook.io/docs/rook/v1.10/Helm-Charts/operator-chart/
464 https://rook.io/docs/rook/v1.10/Helm-Charts/ceph-cluster-chart/

5.2.5.3 Monitoring

Provides monitoring capabilities by collecting metrics, including cost metrics, for Kubernetes and applications deployed on managed clusters. Also provides visualization of metrics and evaluates rule expressions to trigger alerts when specific conditions are observed.

- kubecost[465]: provides real-time cost visibility and insights for teams using Kubernetes, helping you continuously reduce your cloud costs.
- kubernetes-dashboard[466]: A general purpose, web-based UI for Kubernetes clusters. It allows users to manage applications running in the cluster, troubleshoot them and manage the cluster itself.
- kube-prometheus-stack[467]: A stack of applications that collect metrics and provide visualization and alerting capabilities.
  NOTE: Prometheus[468], Prometheus Alertmanager[469] and Grafana[470] are included in the bundled installation.
- nvidia-gpu-operator[471]: The NVIDIA GPU Operator manages NVIDIA GPU resources in a Kubernetes cluster and automates tasks related to bootstrapping GPU nodes.
- prometheus-adapter[472]: Provides cluster metrics from Prometheus.

Common Platform Application Name	APP ID
Kubecost	kubecost
Kubernetes Dashboard	kubernetes-dashboard
Full Prometheus Stack	kube-prometheus-stack
Prometheus Adapter	prometheus-adapter
NVIDIA GPU Operator	nvidia-gpu-operator

5.2.5.4 Security

Allows management of security constraints and capabilities for the clusters and users.

---

465 https://kubecost.com/
466 https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/
467 https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack
468 https://prometheus.io/
469 https://prometheus.io/docs/alerting/latest/alertmanager
470 https://grafana.com/
471 https://catalog.ngc.nvidia.com/orgs/nvidia/containers/gpu-operator
472 https://github.com/DirectXMan12/k8s-prometheus-adapter

- gatekeeper[473]: A policy Controller for Kubernetes.

Platform Application	APP ID
Gatekeeper	gatekeeper

5.2.5.5 Single Sign On (SSO)

Group of platform applications that allow enabling SSO on attached clusters. SSO is a centralized system for connecting attached clusters to the centralized authority on the management cluster.

- kube-oidc-proxy[474]: A reverse proxy server that authenticates users using OIDC to Kubernetes API servers where OIDC authentication is not available.
- traefik-forward-auth[475]: Installs a forward authentication application providing Google OAuth based authentication for Traefik.

Platform Application	APP ID
Kube OIDC Proxy	kube-oidc-proxy
Traefik ForwardAuth	traefik-forward-auth

5.2.5.6 Backup

This platform application assists you with backing up and restoring your environment.

- velero[476]: An open source tool for safely backing up and restoring resources in a Kubernetes cluster, performing disaster recovery, and migrating resources and persistent volumes to another Kubernetes cluster.

Platform Application	APP ID
Velero	velero

Review the Workspace Platform Application Defaults and Resource Requirements (see page 124) to ensure that the attached clusters have sufficient resources.

473 https://github.com/open-policy-agent/gatekeeper
474 https://github.com/jetstack/kube-oidc-proxy
475 https://github.com/thomseddon/traefik-forward-auth
476 https://velero.io/

When deploying and upgrading applications, platform applications come as a bundle; they are tested as a single unit, and you must deploy or upgrade them in a single process, for each workspace. This means all clusters in a workspace have the same set and versions of platform applications deployed.

5.2.5.6.1 Related Topics

- Deploy Platform Applications via CLI (see page 667)
- Cluster-scoped Configuration for Existing AppDeployments (see page 682)
- Manage an Application using the UI (see page 656)
- Cluster-scoped Application Configuration from the DKP UI (see page 680)
- Platform Application Dependencies (see page 669)
- Workspace Platform Application Defaults and Resource Requirements (see page 124)

5.2.5.7 Deploy Platform Applications via CLI

This topic describes how to use the CLI to enable an application to deploy to managed and attached clusters in a workspace.

5.2.5.7.1 Related Topics

- **Available Workspace Platform Applications** (see page 124)
- **Customize** an existing AppDeployment (see page 644)
- **Cluster-scoped** configuration via CLI (see page 682)
- **Cluster-scoped** configuration via UI (see page 680)
- Enterprise Upgrade Platform Applications (see page 1722) OR Essential: Upgrade the Essential Cluster and Platform Applications (see page 1748) depending on your license type.

5.2.5.7.2 Prerequisites

Before you begin, you must have:

- A running cluster with Kommander installed (see page 76).
- An existing Kubernetes cluster attached to Kommander (see page 784).
- Determine the name of the workspace where you wish to perform the deployments. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.
- Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace where the cluster is attached:

  ```
  export WORKSPACE_NAMESPACE=<workspace_namespace>
  ```

- Set the `WORKSPACE_NAME` environment variable to the name of the workspace where the cluster is attached:

```
export WORKSPACE_NAME=<workspace_name>
```

From the CLI, you can enable applications to deploy in the workspace. Verify that an application has successfully deployed via the CLI (see page 668).

5.2.5.7.3 Create the `AppDeployment` to Enable the Application

Enable a supported application to deploy to your existing attached or managed cluster with an AppDeployment resource (see page 644).

1. Obtain the **APP ID** and **Version** of the application from the applications table in the latest Release Notes[477]. You will need them it in the `<APP-ID>-<Version>` format, for example, `istio-1.17.2`.
2. Run the following command and define the `--app` flag to specify which platform application and version will be enabled:

```
dkp create appdeployment istio --app istio-1.17.2 --workspace ${WORKSPACE_NAME}
```

- The `--app` flag must match the `APP NAME` from the list of available platform applications.
- Observe that the `dkp create` command must be run with the `WORKSPACE_NAME` instead of the `WORKSPACE_NAMESPACE` flag.

This instructs Kommander to create and deploy the `AppDeployment` to the `KommanderClusters` in the specified `WORKSPACE_NAME`.

5.2.5.7.4 Verify Applications

The applications are now enabled. Connect to the attached cluster and watch the `HelmReleases` to verify the deployment. In this example, we are checking if `istio` got deployed correctly:

477 https://d2iq.atlassian.net/wiki/pages/createpage.action?fromPageId=29919761&linkCreation=true&spaceKey=DENT&title=DKP+2.6.0+Components+and+Applications

```
kubectl get helmreleases istio -n ${WORKSPACE_NAMESPACE} -w
```

You should eventually see the `HelmRelease` marked as `Ready`:

```
NAMESPACE                NAME        READY   STATUS                                AGE
workspace-test-vjsfq     istio       True    Release reconciliation succeeded      7m3s
```

Some supported applications have dependencies on other applications. See Workspace Platform Application Dependencies (see page 669) for that table.

5.2.5.8 Platform Application Dependencies

5.2.5.8.1 Dependencies between workspace applications

Platform applications that are deployed to a workspace's attached clusters can depend on each other. It is important to note these dependencies when customizing the workspace platform applications to ensure that your applications are properly deployed to the clusters. For more information on how to customize workspace platform applications, see Workspace Platform Applications (see page 662).

When deploying or troubleshooting platform applications, it helps to understand how platform applications interact and may require other platform applications as dependencies.

If a platform application's dependency does not successfully deploy, the platform application requiring that dependency does not successfully deploy.

The following sections detail information about the workspace platform application.

5.2.5.8.2 Foundational Applications

Provides the foundation for all platform application capabilities and deployments on managed clusters. These applications must be enabled for any platform applications to work properly.

The foundational applications are comprised of the following platform application:

- cert-manager[478]: Automates TLS certificate management and issuance.
- reloader[479]: A controller that watches changes on ConfigMaps and Secrets, and automatically triggers updates on the dependent applications.
- traefik[480]: Provides an HTTP reverse proxy and load balancer. Requires cert-manager and reloader.

478 https://cert-manager.io/docs
479 https://github.com/stakater/Reloader
480 https://traefik.io/

Platform Application	Required Dependencies
cert-manager	-
reloader	-
traefik	cert-manager, reloader

5.2.5.8.3 Logging

Collects logs over time from Kubernetes and applications deployed on managed clusters. Also provides the ability to visualize and query the aggregated logs.

- fluent-bit[481]: Open source and multi-platform log processor tool which aims to be a generic Swiss knife for logs processing and distribution.
- grafana-logging[482]: Logging dashboard used to view logs aggregated to Grafana Loki.
- grafana-loki[483]: A horizontally-scalable, highly-available, multi-tenant log aggregation system inspired by Prometheus.
- logging-operator[484]: Automates the deployment and configuration of a Kubernetes logging pipeline.
- rook-ceph[485] and rook-ceph-cluster[486]: A Kubernetes-native high performance object store with an S3-compatible API that supports deploying into private and public cloud infrastructures.

Platform Application	Required Dependencies	Optional Dependencies
fluent-bit	-	-
grafana-logging	grafana-loki	-
grafana-loki	rook-ceph-cluster	-
logging-operator	-	-
rook-ceph	-	-

481 https://docs.fluentbit.io/manual/
482 https://grafana.com/oss/grafana/
483 https://grafana.com/oss/loki/
484 https://banzaicloud.com/docs/one-eye/logging-operator/
485 https://rook.io/docs/rook/v1.10/Helm-Charts/operator-chart/
486 https://rook.io/docs/rook/v1.10/Helm-Charts/ceph-cluster-chart/

Platform Application	Required Dependencies	Optional Dependencies
rook-ceph-cluster	rook-ceph	kube-prometheus-stack Users can override the configuration to remove the dependency, as needed.

5.2.5.8.4 Monitoring

Provides monitoring capabilities by collecting metrics, including cost metrics, for Kubernetes and applications deployed on managed clusters. Also provides visualization of metrics and evaluates rule expressions to trigger alerts when specific conditions are observed.

- kubecost[487]: provides real-time cost visibility and insights for teams using Kubernetes, helping you continuously reduce your cloud costs.
- kubernetes-dashboard[488]: A general purpose, web-based UI for Kubernetes clusters. It allows users to manage applications running in the cluster, troubleshoot them and manage the cluster itself.
- kube-prometheus-stack[489]: A stack of applications that collect metrics and provide visualization and alerting capabilities.
  NOTE: Prometheus[490], Prometheus Alertmanager[491] and Grafana[492] are included in the bundled installation.
- nvidia-gpu-operator[493]: The NVIDIA GPU Operator manages NVIDIA GPU resources in a Kubernetes cluster and automates tasks related to bootstrapping GPU nodes.
- prometheus-adapter[494]: Provides cluster metrics from Prometheus.

Platform Application	Required Dependencies
kubecost	-
kubernetes-dashboard	-

487 https://kubecost.com/
488 https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/
489 https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack
490 https://prometheus.io/
491 https://prometheus.io/docs/alerting/latest/alertmanager
492 https://grafana.com/
493 https://catalog.ngc.nvidia.com/orgs/nvidia/containers/gpu-operator
494 https://github.com/DirectXMan12/k8s-prometheus-adapter

Platform Application	Required Dependencies
kube-prometheus-stack	-
prometheus-adapter	kube-prometheus-stack
nvidia-gpu-operator	-

5.2.5.8.5 Security

Allows management of security constraints and capabilities for the clusters and users.

- gatekeeper[495]: A policy Controller for Kubernetes.

Platform Application	Required Dependencies
gatekeeper	-

5.2.5.8.6 Single Sign On (SSO)

Group of platform applications that allow enabling SSO on attached clusters. SSO is a centralized system for connecting attached clusters to the centralized authority on the management cluster.

- kube-oidc-proxy[496]: A reverse proxy server that authenticates users using OIDC to Kubernetes API servers where OIDC authentication is not available.
- traefik-forward-auth[497]: Installs a forward authentication application providing Google OAuth based authentication for Traefik.

Platform Application	Required Dependencies
kube-oidc-proxy	cert-manager, traefik
traefik-forward-auth	traefik

495 https://github.com/open-policy-agent/gatekeeper
496 https://github.com/jetstack/kube-oidc-proxy
497 https://github.com/thomseddon/traefik-forward-auth

5.2.5.8.7 Backup

This platform application assists you with backing up and restoring your environment.

- velero[498]: An open source tool for safely backing up and restoring resources in a Kubernetes cluster, performing disaster recovery, and migrating resources and persistent volumes to another Kubernetes cluster.

Platform Application	Dependencies
velero	rook-ceph-cluster This is an optional dependency. Users can override the config to remove the dependency, as needed.

5.2.5.8.8 Service Mesh

Allows deploying service mesh on clusters, enabling the management of microservices in cloud-native applications. Service mesh can provide a number of benefits, such as providing observability into communications, providing secure connections, or automating retries and backoff for failed requests.

- istio[499]: Addresses the challenges developers and operators face with a distributed or microservices architecture.
- jaeger[500]: A distributed tracing system used for monitoring and troubleshooting microservices-based distributed systems.
- kiali[501]: A management console for an Istio-based service mesh. It provides dashboards, observability, and lets you operate your mesh with robust configuration and validation capabilities.

Platform Application	Required Dependencies	Optional Dependencies
istio	kube-prometheus-stack	-
jaeger	istio	-

498 https://velero.io/
499 https://istio.io/latest/about/service-mesh/
500 https://www.jaegertracing.io/
501 https://kiali.io/

Platform Application	Required Dependencies	Optional Dependencies
kiali	istio	jaeger (optional for monitoring purposes)
knative	istio	-

5.2.5.8.9 DKP Insights

Analyses your cluster's health, detects current and future anomalies, and creates alerts that offer root cause analysis and recommended solutions.

- dkp-insights: Also called **DKP Insights Engine**, collects events and metrics on your cluster and uses rule-based heuristics to detect potential anomalies of varying criticality.
- dkp-insights-management: Also called **DKP Insights Management**, collects anomaly alerts from one or several clusters and displays root cause analysis and recommended solutions in the UI.

Platform Application	Required Dependencies	Optional Dependencies
dkp-insights	Management/Essential Cluster • dkp-insights-management Workspace • rook-ceph • rook-ceph-cluster • kube-prometheus-stack	-
dkp-insights-management	kubefed	-

5.2.5.8.10 DKP AI Navigator Cluster Info Agent

Coupled with the AI Navigator, it analyses your cluster's data to include live information on queries made through the AI Navigator chatbot.

- `ai-navigator-info-api` : This is the collector of the application's API service, which performs all data abstraction data structuring services. This component is enabled by default and included in the AI Navigator.
- `ai-navigator-info-agent` : After manually enabling this platform application (see page 1987), the agent starts collecting Essential or Management cluster (see page 79) data and injecting it into the Cluster Info Agent database.

Platform Application	Required Dependencies	Optional Dependencies
ai-navigator-info-agent	On the Management/Essential Cluster • ai-navigator-info-api (included in ai-navigator-app)	-

5.2.5.8.11 Related Information

- Kommander security architecture (see page 964)
- Traefik Ingress controller (see page 964)
- Logging and audits (see page 928)

5.2.5.9 Workspace Platform Application Resource Requirements

Refer to Workspace Platform Application Defaults and Resource Requirements (see page 124) for a list of all platform applications, their default deployment configuration, required resources, and storage minimums.

5.2.6 Setting Priority Classes in DKP Applications

In DKP, your workloads can be prioritized to ensure that your critical components stay running in any situation.

Priority Classes can be set for any application in DKP, including Platform Applications, Catalog applications, and even your own Custom Applications.

By default, the priority classes of Platform Applications are set by DKP.

See the following pages for more information about the default priority classes for DKP applications:

- Workspace Platform Application Defaults and Resource Requirements (see page 124)
- Management Cluster Application Requirements (see page 122)
- Project Platform Application Configuration Requirements (see page 724)

This page gives instructions on how you can override the default priority class of any application in DKP to a different one.

5.2.6.1 DKP Priority Classes

Here are the priority classes that are available in DKP:

Class	Value Name	Value	Description
DKP High	`dkp-high-priority`	100001000	This is the priority class that is used for high priority DKP workloads.
DKP Critical	`dkp-critical-priority`	100002000	This is the highest priority class that is used for critical priority DKP workloads.

5.2.6.2 Prerequisites

Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace where the cluster is attached:

```
export WORKSPACE_NAMESPACE=<your_workspace_namespace>
```

You are now able to copy the following commands without having to replace the placeholder with your workspace namespace every time you run a command.

5.2.6.3 Set the Priority Class

Follow these steps.

Keep in mind that the overrides for each application appears differently and is dependent on how the application's helm chart values are configured.

For more information about the helm chart values used in the DKP Platform Applications, see DKP 2.7.0 Components and Applications (see page 1950).

Generally speaking, performing a search for the `priorityClassName` field allows you to find out how you can set the priority class for a component.

In the example below which uses the helm chart values in Grafana Loki, the referenced `priorityClassName` field is nested under the `ingester` component. That said, if you reference the Grafana Loki helm chart values[502], you'll see that the priority class can be set for several other components, including `distributor`, `ruler`, and on a `global` level.

502 https://github.com/grafana/helm-charts/blob/main/charts/loki-distributed/values.yaml

1. Create a `ConfigMap` with custom priority class configuration values for Grafana Loki. The following example sets the priority class of ingester component to the DKP critical priority class:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: grafana-loki-overrides
data:
  values.yaml: |
    ingester:
      priorityClassName: dkp-critical-priority
EOF
```

2. Edit the `grafana-loki` AppDeployment to set the value of `spec.configOverrides.name` to `grafana-loki-overrides` (Refer to Deploy a service with a custom configuration (see page 0) for more information).

```
dkp edit appdeployment -n ${WORKSPACE_NAMESPACE} grafana-loki
```

After your editing is complete, the AppDeployment resembles this example:

```
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: grafana-loki
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: grafana-loki-0.69.16
    kind: ClusterApp
  configOverrides:
    name: grafana-loki-overrides
```

3. It will take a few minutes to reconcile, but you can check the ingester pod's priority class after reconciling:

```
kubectl get pods -n ${WORKSPACE_NAMESPACE} -o custom-
columns=NAME:.metadata.name,PRIORITY:.spec.priorityClassName,PRIORITY:.spec.pri
ority |grep ingester
```

It will show something like this:

```
NAME                                                              PRIORITY
PRIORITY
grafana-loki-loki-distributed-ingester-0                          dkp-critical-
priority     100002000
```

For more information about priority classes, see https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/

5.3 Workspaces

Enterprise Gov Advanced

Allow teams or tenants to manage their own clusters using workspaces

Workspaces are a logical grouping of clusters that maintain a similar configuration, with certain configurations automatically federated to those clusters. Workspaces give you the flexibility to represent your organization in a way that makes sense for your teams or tenants. For example, you can create workspaces to separate clusters according to departments, products, or business functions. You can also create workspaces as a Managed Service Provider (see page 105) to administrate each of your tenant's environments. With separate workspaces, each team or tenant can manage its own clusters and role-based permissions, while you retain an overall organization-level view of all clusters in operation.

5.3.1 Global / Workspace UI

The UI is designed to be accessible for different roles at different levels:

- **Global**: At the top level, IT administrators manage all clusters across all workspaces.
- **Workspace**: DevOps administrators manage multiple clusters within a workspace.
- **Projects**: DevOps administrators or developers manage configuration and services across multiple clusters.

5.3.2 Default Workspace

To get started immediately, you can use the default workspace deployed in DKP. However, take into account that you cannot move clusters from one workspace to another after creating/attaching them.

5.3.3 Create a Workspace

In DKP, you can create your own Workspaces. The following steps describe this procedure.

1. From the workspace selection dropdown in the top menu bar, select **Create Workspace**.
2. Type a name and description and select **Save**. The workspace is now accessible from the workspace selection drop-down menu.

5.3.4 Add, Edit, and Delete Workspace Annotations and Labels

When creating or editing a workspace, you can use the **Advanced Options** to add, edit, or delete annotations and labels to your workspace. Both the annotations and labels are applied to the workspace namespace.

1. From the top menu bar, select your target workspace.
2. Select the **Actions** drop-down button in the top-right, and select **Edit Workspace**.
3. Type in new **Key** and **Value** labels for your workspace, or edit existing **Key** and **Value** labels.

> Labels that are added to a workspace, are also applied to the `kommanderclusters` object as well as to all the clusters in the workspace.

5.3.5 Delete a Workspace

In DKP, you can delete existing Workspaces. The following steps describe this procedure.

> Workspaces can only be deleted if all the clusters in the workspace have been deleted or detached.

1. From the top menu bar, select **Global**.
2. From the sidebar menu, select **Workspaces**.
3. Select the three-dot button to the right of the workspace you want to delete, and then select **Delete**.
4. Confirm deleting the Workspace in the **Delete Workspace** dialog box.

The following procedures are supported for workspaces:

- Application Deployment using the CLI (see page 667)
- Platform Applications (see page 662)
- Platform Application Dependencies (see page 669)
- Workspace Platform Application Defaults and Resource Requirements (see page 124)

5.3.6 Workspace Applications

Enterprise Gov Advanced

This section documents the applications and application types that you can use with DKP.

Application types are:

- Catalog Applications (see page 691) are either pre-packaged applications from the D2iQ Application Catalog, or custom applications that you maintain for your teams or organization.
- Platform Applications (see page 662) are applications integrated into DKP.

- Cluster-scoped Application Configuration from the DKP UI (see page 680)
- Cluster-scoped Configuration for Existing AppDeployments (see page 682)

5.3.6.1 Cluster-scoped Application Configuration from the DKP UI

Enterprise Gov Advanced

When you enable an application for a workspace, you deploy this application to all clusters within that workspace. You can also choose to enable or customize an application on certain clusters within a workspace.

This functionality allows you to use DKP in a multi-cluster scenario without restricting the management of multiple clusters from a single workspace.

NOTE: DKP Essential users are only be able to configure and deploy applications to a single cluster within a workspace. Selecting an application to deploy to a cluster skips cluster selection and takes you directly to the workspace configuration overrides page.

5.3.6.1.1 Prerequisites

Ensure that you've provisioned or attached clusters in one of the following environments:

- Amazon Web Services (AWS) (see page 1225)
- Amazon Elastic Kubernetes Service (EKS) (see page 1278)
- Microsoft Azure (see page 775)

Refer to the current list of Catalog and Platform Applications:

- Workspace Catalog Applications (see page 691)
- Workspace Platform Applications (see page 124)

5.3.6.1.2 Enable a Cluster-scoped Application

Navigate to the workspace containing the clusters you want to deploy to by selecting the appropriate workspace name from the drop-down menu at the top of the DKP dashboard.

1. Select **Applications** from the left navigation bar and find the application you want to deploy to the cluster.
2. Select the the three-dot menu in the desired application's tile and select **Enable** to reach the application enablement page.

NOTE: The application enablement page can also be reached by selecting **View Details** from the three-dot menu, and then selecting **Enable** from the application's details page.

3. Select the cluster(s) that you're deploying the application to. The available clusters can be sorted by **Name**, **Type**, **Provider** and any **Labels** you've added.
4. Deploy the application to the clusters by selecting **Enable** in the top right corner of the application enablement page.

You are automatically redirected to either the **Applications** or **View Details** page. To view the application enabled in your chosen cluster, navigate to the **Clusters** page on the left navigation bar. The application appears in the **Applications** pane of the appropriate cluster.

NOTE: Once you enable an application at the workspace level, DKP automatically enables that app on any other cluster you create or attach.

5.3.6.1.3 Configure a Cluster-scoped Application

For scenarios where applications require different configurations on a per-cluster basis, navigate to the **Applications** page and select **Edit** from the three-dot menu of the appropriate application to return to the application enablement page.

1. Select the cluster(s) that you're deploying the application to. The available clusters can be sorted by **Name**, **Type**, **Provider** and any **Labels** you've added.
2. Select the **Configuration** tab.
3. The **Configuration** tab contains two separate types of code editors, where you can enter your specified overrides and configurations:
   Workspace Application Configuration: A workspace-level code editor that applies all configurations and overrides to the entirety of the workspace and its clusters for this application.
   Cluster Application Configuration Override: A cluster-scoped code editor that applies configurations and overrides to the cluster specified. These customizations will merge with the workspace application configuration. If there is no cluster-scoped configuration, the workspace configuration applies.
4. If you already have a configuration to apply in a text or .yaml file, you can upload the file by selecting **Upload File**. If you want to download the displayed set of configurations, select **Download File**.
5. Finish configuring the cluster-scoped applications by selecting **Save** in the top right corner of the application enablement page.

You are automatically redirected to either the **Applications** or **View Details** page. To view the custom configurations of the application in the cluster, select the **Configurations** tab on the details page of the application.

NOTE: Editing is disabled in the code boxes displayed in the application's details page. To edit the configuration, select the **Edit** button in the top right of the page and repeat the steps in this section.

5.3.6.1.4 Remove a Cluster-scoped Application

Navigate to the cluster you've deployed your applications to by selecting **Clusters** from the left navigation bar.

1. Click on the Applications tab.
2. Select the three-dot menu in the desired application's tile and select **Uninstall.**
3. A prompt appears to confirm your decision to uninstall the application. Follow the instructions in the prompt and select **Uninstall.**
4. Refresh the page to confirm that the application has been removed from the cluster.

This process only removes the application from the specific cluster you've navigated to. To remove this application from other clusters, navigate to the **Clusters** page and repeat the process.

5.3.6.2 Cluster-scoped Configuration for Existing AppDeployments

Enterprise Gov Advanced

This topic describes how to enable cluster-scoped configuration of applications for **existing** `AppDeployments`. For more information on how to **create** `AppDeployments`, refer to the Application Deployment (see page 1849) section.

5.3.6.2.1 Enable or Disable an App per Cluster and Define a Custom Configuration

Edit the `AppDeployment` resource to modify the configuration of an application per cluster.

- Prerequisites (per-cluster application configuration) (see page 683)
- Enable an Application per Cluster (see page 683)
- Customizations per Cluster (see page 685)
- Verify the Current Application Configuration (see page 691)

5.3.6.2.2 For Better Understanding

When you enable an application for a workspace, you deploy this application to all clusters within that workspace. You can also choose to enable or customize an application on certain clusters within a workspace. This functionality allows you to use DKP in a multi-cluster scenario without restricting the management of multiple clusters from a single workspace.

Your DKP cluster comes bundled with a set of default application configurations. If you want to override the default configuration of your applications, you can define workspace `configOverrides` on top of the default workspace configuration. And if you want to further customize your workplace by enabling applications on a per-cluster basis or by defining per-cluster customizations, you can create and apply `clusterConfigOverrides`.

The cluster-scoped enablement and customization of applications is an **Enterprise-only** feature, which allows the configuration of all workspace Platform (see page 662), DKP catalog (see page 691) and Custom Applications (see page 702) via the **CLI** in your managed and attached clusters regardless of your environment setup (air-gapped or non-air-gapped). This capability is not provided for project applications.

5.3.6.2.3 Prerequisites (per-cluster application configuration)

Enterprise Gov Advanced

- Any application you wish to enable or customize at a cluster level, first needs to be enabled at the workspace-level via an `AppDeployment` (platform application deployment (see page 667), workspace catalog application deployment (see page 697)).
- For custom configurations, you have created a ConfigMap (see page 0) with all the required `spec` fields for each customization you would like to add to an application in a cluster. You can apply a `ConfigMap` to several clusters, or create a `ConfigMap` for each cluster, but the `ConfigMap` object must exist in the Management cluster.
- Determine the name of the workspace where you wish to perform the deployments. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.
- Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace where the cluster is attached:

  ```
  export WORKSPACE_NAMESPACE=<workspace_namespace>
  ```

- Set the `WORKSPACE_NAME` environment variable to the name of the workspace where the cluster is attached:

  ```
  export WORKSPACE_NAME=<workspace_name>
  ```

5.3.6.2.4 Enable an Application per Cluster

Enterprise Gov Advanced

5.3.6.2.4.1 Prerequisites

Ensure you have reviewed the prerequisites (see page 683) section.

5.3.6.2.4.2 Enable an Application per Cluster *for the First Time*

When you enable an application on a workspace, it is deployed to all clusters in the workspace by default. If you would like to only deploy it to a subset of clusters when enabling it on a workspace for the first time, you can follow the steps below:

1. Create an `AppDeployment` for your application, selecting a subset of clusters within the workspace to enable it on. You can use the `dkp get clusters --workspace ${WORKSPACE_NAME}` command to see the list of clusters in the workspace.
   The following snippet is an example. Replace the application name, version, workspace name and cluster names according to your requirements. See Components and Applications[503] for compatible versions.

   ```
   dkp create appdeployment kube-prometheus-stack --app kube-prometheus-
   stack-46.8.0 --workspace ${WORKSPACE_NAME} --clusters attached-
   cluster1,attached-cluster2
   ```

2. **Optional:** Check the current status (see page 691) of the `AppDeployment` to see the names of the clusters where the application is currently enabled.

5.3.6.2.4.3 Enable or Disable an Application per Cluster *after it has been enabled at the workspace level*

You can enable or disable applications at any time. After you have enabled the application at the workspace level (see page 0), the `spec.clusterSelector` field (see page 1795) populates.

> For clusters that are newly attached into the workspace, all applications enabled for the workspace are automatically enabled on and deployed to the new clusters.

If you want to see on what clusters your application is currently deployed, refer to the Print and Review the Current State of your AppDeployment (see page 646) documentation.

1. Edit the `AppDeployment` YAML by adding or removing the names of the clusters where you want to enable your application in the `clusterSelector` section:

503 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/214630420/DKP+2.6.0+Components+and+Applications#applications

The following snippet is an example. Replace the application name, version, workspace name and cluster names according to your requirements. See Components and Applications[504] for the compatible versions.

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: kube-prometheus-stack-46.8.0
    kind: ClusterApp
  clusterSelector:
    matchExpressions:
    - key: kommander.d2iq.io/cluster-name
      operator: In
      values:
      - attached-cluster1
      - attached-cluster3-new
EOF
```

5.3.6.2.4.4 Verify the Current Configuration of your Application

Refer to the Verify applications help (see page 0) to connect to the managed or attached cluster and check the status of the deployments.

If you want to know how the `AppDeployment` resource is currently configured, refer to the Print and review the state of your AppDeployment (see page 646) section.

5.3.6.2.5 Customizations per Cluster

Enable or disable a customization per cluster.

- Customize an Application per Cluster (see page 685)
- Disable a Custom Configuration of an Application for a Cluster (see page 688)

5.3.6.2.5.1 Customize an Application per Cluster

Enterprise Gov Advanced

Prerequisites

504 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/214630420/DKP+2.6.0+Components+and+Applications#applications

Ensure you have reviewed the prerequisites (see page 683) section.

Enable a Custom Configuration of an Application for a Cluster

You can customize the application for each cluster occurrence of said application. If you want to customize the application for a cluster that is **not yet attached**, refer to the instructions below, so the application is deployed with the custom configuration during attachment.
To enable per-cluster customizations:

1. Reference the name of the `ConfigMap` to be applied per cluster in the `spec.clusterConfigOverrides` fields. In this example, you have three different customizations specified in three different `ConfigMaps` for three different clusters in one workspace:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: kube-prometheus-stack-46.8.1
    kind: ClusterApp
  clusterSelector:
    matchExpressions:
    - key: kommander.d2iq.io/cluster-name
      operator: In
      values:
      - attached-cluster1
      - attached-cluster2
      - attached-cluster3-new
  clusterConfigOverrides:
    - configMapName: kps-cluster1-overrides
      clusterSelector:
        matchExpressions:
        - key: kommander.d2iq.io/cluster-name
          operator: In
          values:
          - attached-cluster1
    - configMapName: kps-cluster2-overrides
      clusterSelector:
        matchExpressions:
        - key: kommander.d2iq.io/cluster-name
          operator: In
          values:
          - attached-cluster2
    - configMapName: kps-cluster3-overrides
      clusterSelector:
        matchExpressions:
```

```
        - key: kommander.d2iq.io/cluster-name
          operator: In
          values:
          - attached-cluster3-new
EOF
```

2. If you have not done so yet, create the ConfigMaps (see page 0) referenced in each `clusterConfigOverrides` entry.

- The changes are applied only if the YAML file has a valid syntax.
- Set up only one cluster override `ConfigMap` per cluster. If there are several `ConfigMaps` configured for a cluster, only one will be applied.
- Cluster override `ConfigMaps` must be created on the Management cluster.

Enable a Custom Configuration of an Application for a Cluster at Attachment

You can customize the application configuration for a cluster prior to its attachment, so that the application is deployed with this custom configuration on attachment. This is preferable, if you do not want to redeploy the application with an updated configuration after it has been initially installed, which may cause downtime.

To enable per-cluster customizations, follow these steps **before attaching the cluster**:

1. Set the `CLUSTER_NAME` environment variable to the cluster name that you will give your to-be-attached cluster:

```
export CLUSTER_NAME=<your_attached_cluster_name>
```

2. Reference the name of the `ConfigMap` you want to apply to this cluster in the `spec.clusterConfigOverrides` fields. **You do not need to update the `spec.clusterSelector` field**. In this example, you have the `kps-cluster1-overrides` customization specified for `attached-cluster-1` and a different customization (in `kps-your-attached-cluster-overrides` `ConfigMap`) for your to-be-attached cluster:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
```

```
  appRef:
    name: kube-prometheus-stack-46.8.1
    kind: ClusterApp
  clusterSelector:
    matchExpressions:
    - key: kommander.d2iq.io/cluster-name
      operator: In
      values:
      - attached-cluster1
  clusterConfigOverrides:
    - configMapName: kps-cluster1-overrides
      clusterSelector:
        matchExpressions:
        - key: kommander.d2iq.io/cluster-name
          operator: In
          values:
          - attached-cluster1
    - configMapName: kps-your-attached-cluster-overrides
      clusterSelector:
        matchExpressions:
        - key: kommander.d2iq.io/cluster-name
          operator: In
          values:
          - ${CLUSTER_NAME}
EOF
```

3. If you have not done so yet, create the ConfigMap (see page 644) referenced for your to-be-attached cluster.

- The changes are applied only if the YAML file has a valid syntax.
- Cluster override `ConfigMaps` must be created on the Management cluster.

Verify the Current Configuration of your Application

Refer to the Verify applications help (see page 0) to connect to the managed or attached cluster and check the status of the deployments.

If you want to know how the `AppDeployment` resource is currently configured, refer to the Print and review the state of your AppDeployment (see page 646) section.

5.3.6.2.5.2 Disable a Custom Configuration of an Application for a Cluster

Enabled customizations are defined in a `ConfigMap`, which, in turn, is referenced in the `spec.clusterConfigOverrides` object of your `AppDeployment`.

1. Review your current configuration to establish what you want to remove:

   ```
   kubectl get appdeployment -n ${WORKSPACE_NAMESPACE} kube-prometheus-stack -o
   yaml
   ```

   The output looks similar to this:

   ```
   apiVersion: apps.kommander.d2iq.io/v1alpha3
   kind: AppDeployment
   metadata:
     name: kube-prometheus-stack
     namespace: ${WORKSPACE_NAMESPACE}
   spec:
     appRef:
       name: kube-prometheus-stack-46.8.0
       kind: ClusterApp
     configOverrides:
       name: kube-prometheus-stack-overrides-attached
     clusterSelector:
       matchExpressions:
       - key: kommander.d2iq.io/cluster-name
         operator: In
         values:
         - attached-cluster1
         - attached-cluster2
     clusterConfigOverrides:
       - configMapName: kps-cluster1-overrides
         clusterSelector:
           matchExpressions:
           - key: kommander.d2iq.io/cluster-name
             operator: In
             values:
             - attached-cluster1
       - configMapName: kps-cluster2-overrides
         clusterSelector:
           matchExpressions:
           - key: kommander.d2iq.io/cluster-name
             operator: In
             values:
             - attached-cluster2
   ```

   Here you can see that `kube-prometheus-stack` has been enabled for the `attached-cluster1` and `attached-cluster2`. There is also a custom configuration for each of the clusters: `kps-cluster1-overrides` and `kps-cluster2-overrides`.

2. Edit the file by deleting the `configMapName` entry of the cluster for which you would like to delete the customization. This is located under the `clusterConfigOverrides`:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    kind: ClusterApp
    name: kube-prometheus-stack-46.8.0
  configOverrides:
    name: kube-prometheus-stack-ws-overrides
  clusterSelector:
    matchExpressions:
      - key: kommander.d2iq.io/cluster-name
        operator: In
        values:
        - attached-cluster1
  clusterConfigOverrides:
    - configMapName: kps-cluster1-overrides
      clusterSelector:
        matchExpressions:
        - key: kommander.d2iq.io/cluster-name
          operator: In
          values:
          - attached-cluster1
EOF
```

Compare steps one and two for a reference of how an entry should be deleted.

3. Before deleting a `ConfigMap` that contains your customization, **ensure you will NOT require it at a later time**. It is not possible to restore a deleted `ConfigMap`.
   If you choose to delete it, run:

```
kubectl delete configmap <name_configmap> -n ${WORKSPACE_NAMESPACE}
```

It is **NOT** possible to delete a `ConfigMap` that is being actively used and referenced in the `configOverride` of any `AppDeployment`.

Verify the Current Configuration of your Application

Refer to the Verify applications help (see page 0) to connect to the managed or attached cluster and check the status of the deployments.

If you want to know how the `AppDeployment` resource is currently configured, refer to the Print and review the state of your AppDeployment (see page 646) section.

5.3.6.2.6 Verify the Current Application Configuration

Enterprise Gov Advanced

5.3.6.2.6.1 Verify the Current Configuration of your Application

Refer to the Verify applications help (see page 0) to connect to the managed or attached cluster and check the status of the deployments.

If you want to know how the `AppDeployment` resource is currently configured, refer to the Print and review the state of your AppDeployment (see page 646) section.

5.3.7 Workspace Catalog Applications

Enterprise Gov Advanced

Catalog applications are any third-party or open source applications that appear in the Catalog. These applications are deployed to be used for customer workloads.

D2iQ provides DKP Catalog Applications (see page 691) for use in your environment.

- Workspace DKP Catalog Applications (see page 691)
- Deployment of Catalog Applications in Workspaces (see page 697)
- Workspace Catalog Application Upgrades (see page 700)
- Custom Applications (see page 702)

5.3.7.1 Workspace DKP Catalog Applications

Enterprise Gov Advanced

Catalog applications are applications provided by D2iQ for use in your environment.

5.3.7.1.1 Prerequisites

- Ensure your clusters run on a supported Kubernetes version for this release of DKP[505], and that this Kubernetes version is also compatible with your catalog application version (see page 692).
- For customers with an **Enterprise license** (see page 89) and a multi-cluster environment, we recommend keeping all clusters on the same Kubernetes version. This ensures your DKP catalog application can run on all clusters in a given workspace.

505 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/150765792/DKP+2.5.0+Supported+Kubernetes+Versions

5.3.7.1.1.1 Install the DKP Catalog via the CLI

Follow these steps to install the DKP catalog from the CLI.

1. Refer to Install Kommander in an Air-gapped Environment (see page 1533), if you are running in air-gapped environment.
2. Set the `WORKSPACE_NAMESPACE` environment variable to the name of your workspace's namespace:

   ```
   export WORKSPACE_NAMESPACE=<workspace namespace>
   ```

3. Create the `GitRepository`:

   ```
   kubectl apply -f - <<EOF
   apiVersion: source.toolkit.fluxcd.io/v1
   kind: GitRepository
   metadata:
     name: dkp-catalog-applications
     namespace: ${WORKSPACE_NAMESPACE}
     labels:
       kommander.d2iq.io/gitapps-gitrepository-type: catalog
       kommander.d2iq.io/gitrepository-type: catalog
   spec:
     interval: 1m0s
     ref:
       tag: v2.8.0
     timeout: 20s
     url: https://github.com/mesosphere/dkp-catalog-applications
   EOF
   ```

4. Verify that you can see the DKP workspace catalog `Apps` available in the UI (in the Applications section in said workspace), and in the CLI, using `kubectl`:

   ```
   kubectl get apps -n ${WORKSPACE_NAMESPACE}
   ```

5.3.7.1.1.2 Compatibility

Ensure that your DKP Catalog application (see page 0) is compatible with:

- The Kubernetes version in all the Managed and Attached clusters of the workspace where you want to install the catalog application.
- The range of Kubernetes versions supported in this release of DKP[506].

506 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/150765792/DKP+2.5.0+Supported+Kubernetes+Versions

If your current Catalog application version is not compatible, upgrade the application to a compatible version.

As of DKP 2.7, only the following versions of Catalog applications are supported; all previous versions (and any other applications previously included in the Catalog) are now deprecated.

Name	App ID	Compatible Kubernetes versions	Application Version
kafka-operator-0.25.1	kafka-operator	1.21-1.27	0.25.1[507]
zookeeper-operator-0.2.16-dkp.1	zookeeper-operator	1.26-1.27	0.2.15[508]

5.3.7.1.2 Kafka in a Workspace

Enterprise Gov Advanced

Information about the Kafka Operator

Apache Kafka[509] is an open-source distributed event streaming platform used for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications. The Kafka Operator[510] is a Kubernetes operator to automate provisioning, management, autoscaling, and operations of Apache Kafka clusters deployed to Kubernetes. It works by watching custom resources, such as `KafkaClusters`, `KafkaUsers`, and `KafkaTopics`, to provision underlying Kubernetes resources (i.e `StatefulSets`) required for a production-ready Kafka Cluster.

5.3.7.1.2.1 Install

Follow these steps to install the Kafka operator in a workspace. This procedure results in a Kafka operator running in a workspace namespace, ready to manage and create Kafka clusters in any project namespaces. See the Kafka in a Project (see page 728) custom resource documentation for more information on creating Kafka clusters.

Only install the Kafka operator once per workspace.

507 https://github.com/banzaicloud/koperator/releases/tag/v0.25.1
508 https://github.com/mesosphere/zookeeper-operator/releases/tag/v0.2.15-d2iq
509 https://kafka.apache.org/
510 https://banzaicloud.com/docs/supertubes/kafka-operator/

1. Follow the generic installation instructions for workspace catalog applications on the Application Deployment (see page 697) page.
2. Within the `AppDeployment`, update the `appRef` to specify the correct `kafka-operator` App. You can find the `appRef.name` by listing the available `Apps` in the workspace namespace:

```
kubectl get apps -n ${WORKSPACE_NAMESPACE}
```

Refer to the Kafka operator Helm Chart documentation[511] for details on custom configuration for the operator.

If you use a custom version of KafkaCluster with `cruise.control`, ensure you use the custom resource image version `2.5.123` in the `.cruiseControlConfig.image` field for both air-gapped and non-air-gapped environments.

Using a custom image for a zookeeper cluster

To avoid the critical CVEs associated with the official kafka image in version `v0.25.1`, a custom image must be specified when creating a zookeeper cluster.

Specify the following custom values in `KafkaCluster` `CRD`:

- `.spec.clusterImage` to `ghcr.io/banzaicloud/kafka:2.13-3.4.1`
- `.spec.cruiseControlConfig.initContainers[*].image` to `ghcr.io/banzaicloud/cruise-control:2.5.123`

5.3.7.1.2.2 Uninstall via the CLI

Uninstalling the Kafka operator does not affect existing `KafkaCluster` deployments. After uninstalling the operator, you must manually remove any remaining Custom Resource Definitions (CRDs) from the operator.

1. Delete all of the deployed Kafka custom resources (see Deleting Kafka Custom Resources (see page 0)).
2. Uninstall a Kafka operator `AppDeployment`:

```
kubectl -n <workspace namespace> delete AppDeployment <name of AppDeployment>
```

3. Remove Kafka CRDs:

511 https://github.com/banzaicloud/koperator/tree/master/charts/kafka-operator#configuration

NOTE: The CRDs are not finalized for deletion until you delete the associated custom resources.

```
kubectl delete crds kafkaclusters.kafka.banzaicloud.io
kafkausers.kafka.banzaicloud.io kafkatopics.kafka.banzaicloud.io
```

5.3.7.1.2.3 Resources

- Kafka Operator Documentation[512]
- Kafka Operator GitHub Repository[513]
- Sample Kafka Operator Custom Resources[514]
- Apache Kafka Documentation[515]

5.3.7.1.3 Zookeeper Operator in a Workspace

Enterprise Gov Advanced

ZooKeeper[516] is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. The ZooKeeper operator[517] is a Kubernetes operator that handles the provisioning and management of ZooKeeper clusters. It works by watching custom resources, such as `ZookeeperClusters`, to provision the underlying Kubernetes resources (`StatefulSets`) required for a production-ready ZooKeeper Cluster.

5.3.7.1.3.1 Install Zookeeper

Follow these steps to install the ZooKeeper operator in a workspace. This procedure results in a ZooKeeper operator running in a workspace namespace, ready to manage and create ZooKeeper clusters in any project namespaces. See the ZooKeeper in a Project (see page 726) custom resource documentation for more information on creating ZooKeeper clusters.

Only install the ZooKeeper operator once per workspace.

1. Follow the generic installation instructions for workspace catalog applications on the Application Deployment (see page 697) page.

512 https://banzaicloud.com/docs/supertubes/kafka-operator/
513 https://github.com/banzaicloud/koperator
514 https://github.com/banzaicloud/koperator/tree/master/config/samples
515 https://kafka.apache.org/documentation/
516 https://zookeeper.apache.org/index.html
517 https://github.com/pravega/zookeeper-operator

2. Within the `AppDeployment`, update the `appRef` to specify the correct `zookeeper-operator` App. You can find the `appRef.name` by listing the available `Apps` in the workspace namespace:

```
kubectl get apps -n ${WORKSPACE_NAMESPACE}
```

For details on custom configuration for the operator, refer to the ZooKeeper operator Helm Chart documentation[518].

Using a image for a zookeeper cluster

To avoid the critical CVEs associated with the official zookeeper image in version `v0.2.15`, a custom image must be specified when creating a zookeeper cluster.

```
apiVersion: "zookeeper.pravega.io/v1beta1"
kind: "ZookeeperCluster"
# ...
spec:
  image:
    repository: ghcr.io/mesosphere/zookeeper
    tag: v0.2.15-d2iq
```

For more information about custom images go to https://d2iq.atlassian.net/wiki/spaces/DENT/pages/356384780/Enterprise+Upgrade+images+used+by+Catalog+Applications#Upgrading-a-Zookeeper-image-reference-in-a-ZookeeperCluster-resource (see page 0).

5.3.7.1.3.2 Uninstall via the CLI

Uninstalling the ZooKeeper operator will not directly affect any running `ZookeeperClusters`. By default, the operator waits for any `ZookeeperClusters` to be deleted before it will fully uninstall (you can set `hooks.delete: true` in the application configuration to disable this behavior). After uninstalling the operator, you need to manually clean up any leftover Custom Resource Definitions (CRDs).

1. Delete all `ZookeeperClusters`, see Deleting ZooKeeper Clusters (see page 728).
2. Uninstall a ZooKeeper operator `AppDeployment`:

```
kubectl -n <workspace namespace> delete AppDeployment <name of AppDeployment>
```

3. Remove ZooKeeper CRDs:

   WARNING: After you remove the CRDs, all deployed ZookeeperClusters will be deleted!

518 https://github.com/pravega/zookeeper-operator/tree/master/charts/zookeeper-operator#configuration

```
kubectl delete crds zookeeperclusters.zookeeper.pravega.io
```

5.3.7.1.3.3 Resources

- ZooKeeper Operator Documentation[519]
- ZooKeeper Cluster Helm Chart[520]
- ZooKeeper Documentation[521]

5.3.7.2 Deployment of Catalog Applications in Workspaces

Enterprise Gov Advanced

Deploy applications to attached clusters using the CLI

This topic describes how to use the CLI to deploy a workspace catalog application to attached clusters within a workspace. To deploy an application to selected clusters within a workspace, refer to the Cluster-scoped Configuration (see page 682) section of the documentation.

5.3.7.2.1 Prerequisites

Before you begin, you must have:

- A running cluster with Kommander installed (see page 1558). The cluster should be on a supported Kubernetes version for this release of DKP (see page 1946), and also compatible with the catalog application version (see page 692) you wish to install.
- An Attach an Existing Kubernetes Cluster (see page 784) section of the documentation completed.

Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace the attached cluster exists in:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

After creating a GitRepository, use either the DKP UI or the CLI to enable your catalog applications.

From within a workspace, you can enable applications to deploy. Verify that an application has successfully deployed via the CLI (see page 700).

519 https://github.com/pravega/zookeeper-operator
520 https://github.com/pravega/zookeeper-operator/tree/master/charts/zookeeper
521 https://zookeeper.apache.org/documentation

5.3.7.2.2 Enable (and Customize) the Application using the DKP UI

Follow these steps to enable your catalog applications from the DKP UI:

1. **Enterprise only:** from the top menu bar, select your target workspace.
2. Select **Applications** from the sidebar menu to browse the available applications from your configured repositories.
3. Select the three dot button from the bottom-right corner of the desired application card, and then select **Enable**.
4. If available, select a version from the drop-down menu. This drop-down menu will only be visible if there is more than one version.
5. (Optional) If you want to override the default configuration values, copy your customized values into the text editor under **Configure Service** or upload your YAML file that contains the values:

   ```
   someField: someValue
   ```

6. Confirm the details are correct, and then select the **Enable** button.

For all applications, you must provide a display name and an ID which is automatically generated based on what you enter for the display name, unless or until you edit the ID directly. The ID must be compliant with Kubernetes DNS subdomain name validation rules[522].

Alternately, you can use the CLI to enable your catalog applications (see page 0).

5.3.7.2.3 Enable the Application using the CLI

See Workspace Catalog Applications (see page 691) for the list of available applications that you can deploy on the attached cluster.

1. Enable a supported application to deploy to your Attached Cluster (see page 784) with an `AppDeployment` resource.
2. Within the `AppDeployment`, define the `appRef` to specify which `App` to enable:

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: apps.kommander.d2iq.io/v1alpha3
   kind: AppDeployment
   metadata:
     name: kafka-operator
     namespace: ${WORKSPACE_NAMESPACE}
   spec:
     appRef:
       name: kafka-operator-0.25.1
   ```

522 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#dns-subdomain-names

```
    kind: App
EOF
```

- The `appRef.name` must match the app `name` from the list of available catalog applications.
- Create the resource in the workspace you just created, which instructs Kommander to deploy the `AppDeployment` to the `KommanderCluster`s in the same workspace.

5.3.7.2.4 Enable an Application with a Custom Configuration using the CLI

Follow these steps:

Provide the name of a `ConfigMap` in the `AppDeployment`, which provides custom configuration on top of the default configuration:

1.
```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kafka-operator
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: kafka-operator-0.25.1
    kind: App
  configOverrides:
    name: kafka-operator-overrides
EOF
```

2. Create the `ConfigMap` with the name provided in the step above, with the custom configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: kafka-operator-overrides
data:
  values.yaml: |
    operator:
      verboseLogging: true
```

```
EOF
```

Kommander waits for the `ConfigMap` to be present before deploying the `AppDeployment` to the managed or attached clusters.

5.3.7.2.5 Verify Applications

The applications are now enabled. Connect to the attached cluster and check the `HelmReleases` to verify the deployment:

```
kubectl get helmreleases -n ${WORKSPACE_NAMESPACE}
```

The output looks similar to this:

```
NAMESPACE              NAME              READY   STATUS
AGE
workspace-test-vjsfq   kafka-operator    True    Release reconciliation succeeded
7m3s
```

5.3.7.3 Workspace Catalog Application Upgrades

Enterprise Gov Advanced

Upgrade catalog applications using the CLI and UI

5.3.7.3.1 Prerequisites

See Upgrade Prerequisites (see page 1709).

5.3.7.3.2 Upgrade Catalog Applications

Before upgrading, keep in mind the distinction between Platform applications and Catalog applications. Platform applications are deployed and upgraded as a set for each cluster or workspace. Catalog applications are deployed separately, so that you can deploy and upgrade them individually for each workspace or project.

5.3.7.3.2.1 Upgrade with UI

Follow these steps to upgrade an application from the DKP UI:

1. From the top menu bar, select your target workspace.
2. Select **Applications** from the sidebar menu.

3. Select the three dot button from the bottom-right corner of the desired application tile, and then select **Edit**.
4. Select the **Version** drop-down, and select a new version. This drop-down will only be available if there is a newer version to upgrade to.
5. Select **Save**.

5.3.7.3.2.2 Upgrade with CLI

Please note that the below commands are using the workspace **name** and not namespace. You can retrieve the workspace name by running the command:
`dkp get workspaces`
To view a list of the deployed `apps` to your workspace, you can run the command:
`dkp get appdeployments --workspace=<workspace-name>`

1. To see what app(s) and app versions are available to upgrade, run the following command:
   NOTE: You can reference the app version by going into the app name (e.g. `<APP ID>-<APP VERSION>`)

   ```
   kubectl get apps -n ${WORKSPACE_NAMESPACE}
   ```

   You can also use this command to display the apps and app versions, for example:

   ```
   kubectl get apps -n ${WORKSPACE_NAMESPACE} -o jsonpath='{range .items[*]}
   {@.spec.appId}{"----"}{@.spec.version}{"\n"}{end}'
   ```

   Below is an example of an output that shows the different apps and apps versions.

   ```
   kafka-operator----0.20.0
   kafka-operator----0.20.2
   kafka-operator----0.23.0-dev.0
   kafka-operator----0.25.1
   zookeeper-operator----0.2.13
   zookeeper-operator----0.2.14
   zookeeper-operator----0.2.15
   ```

2. Run the following command to upgrade an application from the DKP CLI:

```
dkp upgrade catalogapp <appdeployment-name> --workspace=<my-workspace-name> --
to-version=<version.number>
```

As an example, the following command upgrades the Kafka Operator application, named `kafka-operator-abc`, in a workspace to version `0.25.1`:

```
dkp upgrade catalogapp kafka-operator-abc --workspace=my-workspace --to-version=0.25.1
```

Platform applications cannot be upgraded on a one-off basis, and must be upgraded in a single process for each workspace. If you attempt to upgrade a platform application with these commands, you receive an error and the application is not upgraded.

5.3.7.4 Custom Applications

Enterprise Gov Advanced

Custom applications are third-party applications you have added to the DKP Catalog.

Custom applications are any third-party applications that are not provided in the DKP Application Catalog. Custom applications can leverage applications from the DKP Catalog or be fully-customized. There is no expectation of support by D2iQ for a Custom application. Custom applications can be deployed on Konvoy clusters or on any D2iQ supported 3rd party Kubernetes distribution.

- Create a Git Repository (see page 702)
- Git Repository Structure (see page 704)
- Workspace Application Metadata (see page 706)
- Enable a Custom Application from the Workspace Catalog (see page 708)

5.3.7.4.1 Create a Git Repository

Enterprise Gov Advanced

Create a Git Repository in the Workspace namespace

Use the CLI to create the GitRepository resource and add a new repository to your Workspace.

1. Refer to Air-gapped Environment (see page 1533) setup instructions section, if you are running in air-gapped environment.
2. Set the `WORKSPACE_NAMESPACE` environment variable to the name of your workspace's namespace:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

3. Adapt the URL of your Git repository:

```
kubectl apply -f - <<EOF
apiVersion: source.toolkit.fluxcd.io/v1
kind: GitRepository
metadata:
  name: example-repo
  namespace: ${WORKSPACE_NAMESPACE}
  labels:
    kommander.d2iq.io/gitapps-gitrepository-type: dkp
    kommander.d2iq.io/gitrepository-type: catalog
spec:
  interval: 1m0s
  ref:
    branch: <your-target-branch-name> # e.g., main
  timeout: 20s
  url: https://github.com/<example-org>/<example-repo>
EOF
```

4. Ensure the status of the `GitRepository` signals a ready state:

```
kubectl get gitrepository example-repo -n ${WORKSPACE_NAMESPACE}
```

The repository commit also displays the ready state:

```
NAME         URL                                                     READY
STATUS                                                          AGE
example-repo https://github.com/example-org/example-repo             True
Fetched revision: master/6c54bd1722604bd03d25dcac7a31c44ff4e03c6a   11m
```

For more information on the GitRepository resource fields and how to make Flux aware of credentials required to access a private Git repository, see the Flux documentation[523].

5.3.7.4.1.1 Troubleshoot

To troubleshoot issues with adding the GitRepository, review the following logs:

```
kubectl -n kommander-flux logs -l app=source-controller
[...]
kubectl -n kommander-flux logs -l app=kustomize-controller
[...]
```

523 https://fluxcd.io/flux/components/source/gitrepositories/#secret-reference

```
kubectl -n kommander-flux logs -l app=helm-controller
[...]
```

5.3.7.4.1.2 Related Information

- Flux[524]
- Flux docs[525]

5.3.7.4.2 Git Repository Structure

Enterprise Gov Advanced

Git repositories must be structured in a specific manner for defined applications to be processed by Kommander.

You must structure your git repository based on the following guidelines, for your applications to be processed properly by Kommander so that they can be deployed.

5.3.7.4.2.1 Git Repository Directory Structure

Use the following basic directory structure for your git repository:

```
├── helm-repositories
│   ├── <helm repository 1>
│   │   ├── kustomization.yaml
│   │   └── <helm repository name>.yaml
│   └── <helm repository 2>
│       ├── kustomization.yaml
│       └── <helm repository name>.yaml
└── services
    ├── <app name>
    │   ├── <app version1> # semantic version of the app helm chart. e.g., 1.2.3
    │   │   ├── defaults
    │   │   │   ├── cm.yaml
    │   │   │   └── kustomization.yaml
    │   │   ├── <app name>.yaml
    │   │   └── kustomization.yaml
    │   ├── <app version2> # another semantic version of the app helm chart. e.g.,
2.3.4
    │   │   ├── defaults
    │   │   │   ├── cm.yaml
    │   │   │   └── kustomization.yaml
    │   │   ├── <app name>.yaml
    │   │   └── kustomization.yaml
    │   └── metadata.yaml
```

524 https://fluxcd.io/
525 https://fluxcd.io/docs

```
    └── <another app name>
    ...
```

- Define applications in the `services/` directory.
- You can define multiple versions of an application, under different directories nested under the `services/<app name>/` directory.
- Define application manifests, such as a HelmRelease[526], under each versioned directory `services/<app name>/<version>/` in `<app name>.yaml` which is listed in the `kustomization.yaml` Kubernetes Kustomization file. For more information, see the Kubernetes Kustomization docs[527].
- Define the default values ConfigMap for `HelmReleases` in the `services/<app name>/<version>/defaults` directory, accompanied by a `kustomization.yaml` Kubernetes Kustomization file pointing to the `ConfigMap` file.
- Define the `metadata.yaml` of each application under the `services/<app name>/` directory. For more information, see the Workspace Application Metadata docs (see page 706).

See the DKP Catalog repository[528] for an example of how to structure custom catalog Git repositories.

5.3.7.4.2.2 Helm Repositories

You must include the `HelmRepository` that is referenced in each `HelmRelease`'s `Chart` spec.

Each `services/<app name>/<version>/kustomization.yaml` must include the path of the YAML file that defines the `HelmRepository`. For example:

```
# services/<app name>/<version>/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - <app name>.yaml
  - ../../../helm-repositories/<helm repository 1>
```

For more information, see the flux documentation about HelmRepositories[529].

5.3.7.4.2.3 Substitution Variables

Some substitution variables[530] are provided.

- `${releaseName}`: For each App deployment, this variable is set to the `AppDeployment` name. Use this variable to prefix the names of any resources that are defined in the application directory in the Git repository so that multiple instances of the same application can be deployed. If you create

526 https://fluxcd.io/docs/components/helm/helmreleases/
527 https://kubectl.docs.kubernetes.io/references/kustomize/kustomization/
528 https://github.com/mesosphere/dkp-catalog-applications
529 https://fluxcd.io/docs/components/source/helmrepositories/
530 https://fluxcd.io/docs/components/kustomize/kustomization/#variable-substitution

resources without using the `releaseName` prefix (or suffix) in the name field, there can be conflicts if the same named resource is created in that same namespace.

- `${releaseNamespace}`: The namespace of the Workspace.
- `${workspaceNamespace}`: The namespace of the Workspace that the Workspace belongs to.

5.3.7.4.2.4 Related Information

- Flux[531]
- Flux docs[532]
- Getting started with Flux guide[533]

5.3.7.4.3 Workspace Application Metadata

Enterprise Gov Advanced

> To display more information about custom applications in the UI, define a metadata.yaml file for each application in the Git repository.

You can define how custom applications display in the DKP UI by defining a `metadata.yaml` file for each application in the git repository. You must define this file at `services/<application>/metadata.yaml` for it to process correctly.

You can define the following fields:

Field	Default	Description
displayName	falls back to App ID	Display name of the application for the UI.
description	“”	Short description, should be a sentence or two, displayed in the UI on the application card.
category	general	1 or more categories for this application. Categories are used to group applications in the UI.

531 https://fluxcd.io/
532 https://fluxcd.io/docs
533 https://fluxcd.io/docs/get-started/

Field	Default	Description
overview		Markdown overview used on the application detail page in the UI.
icon		Base64 encoded icon SVG file used for application logos in the UI.
scope	project	List of scopes, can be set only to project or workspace currently.

None of these fields are required for the application to display in the UI.

Here is an example `metadata.yaml` file:

```
displayName: Prometheus Monitoring Stack
description: Stack of applications that collect metrics and provides visualization
and alerting capabilities. Includes Prometheus, Prometheus Alertmanager and Grafana.
category:
  - monitoring
overview: >
  # Overview
  A stack of applications that collects metrics and provides visualization and
alerting capabilities. Includes Prometheus, Prometheus Alertmanager and Grafana.

  ## Dashboards
  By deploying the Prometheus Monitoring Stack, the following platform applications
and their respective dashboards are deployed. After deployment to clusters in a
workspace, the dashboards are available to access from a respective cluster's detail
page.

  ### Prometheus

  A software application for event monitoring and alerting. It records real-time
metrics in a time series database built using a HTTP pull model, with flexible and
real-time alerting.

  - [Prometheus Documentation - Overview](https://prometheus.io/docs/introduction/
overview/)

  ### Prometheus Alertmanager
  A Prometheus component that enables you to configure and manage alerts sent by the
Prometheus server and to route them to notification, paging, and automation systems.

  - [Prometheus Alertmanager Documentation - Overview](https://prometheus.io/docs/
alerting/latest/alertmanager/)
```

```
  ### Grafana
  A monitoring dashboard from Grafana that can be used to visualize metrics collected
by Prometheus.

  - [Grafana Documentation](https://grafana.com/docs/)
icon:
PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHZpZXdCb3g9IjAgMCAzMDAgMzAwIiBzd
HlsZT0iZW5hYmxlLWJhY2tncm91bmQ6bmV3IDAgMCAzMDAgMzAwIiB4bWw6c3BhY2U9InByZXNlcnZlIj48cG
F0aCBkPSJNMTUwIDUwQzk0LjggNTAgNTAgOTQuOCA1MCAxNTBzNDQuOCAxMDAgMTAwIDEwMCAxMDAtNDQuOCA
xMDAtMTAwUzIwNS4yIDUwIDE1MCA1MHptMCAxODcuMmMtMTUuNyAwLTI4LjUtMTAuNS0yOC41LTIzLjRoNTYu
OWMuMSAxMi45LTEyLjcgMjMuNC0yOC40IDIzLjR6bTQ3LTMxLjJoLTk0di0xN2g5NHYxN3ptLS4zLTI1LjloL
TkzLjRjLS4zLS40LS42LS43LS45LTEuMS05LjYtMTEuNy0xMS45LTE3LjgtMTQuMS0yNCAwLS4yIDExLjcgMi
40IDIwIDQuMyAwIDAgNC4zIDEgMTAuNSAyLjEtNi03LTkuNi0xNi05LjYtMjUuMSAwLTIwIDE1LjQtMzcuNiA
5LjgtNTEuNyA1LjQuNCAxMS4yIDExLjQgMTEuNiAyOC41IDUuNy03LjkgOC4xLTIyLjQgOC4xLTMxLjMgMC05
LjIgNi4xLTE5LjkgMTIuMS0yMC4yLTUuNCA4LjkgMS40IDE2LjUgNy40IDM1LjUgMi4zIDcuMSAyIDE5LjEgM
y43IDI2LjcuNi0xNS44IDMuMy0zOC44IDEzLjMtNDYuNy00LjQgMTAgLjcgMjIuNSA0LjEgMjguNSA1LjYgOS
43IDkgMTcuMSA5IDMxIDAgOS4zLTMuNCAxOC4xLTkuMyAyNSA2LjYtMS4yIDExLjItMi40 IDExLjItMi40bDI
xLjQtNC4yYy4xIDAtMyAxMi44LTE0LjkgMjUuMXoiIHN0eWxlPSJmaWxsOiNmODQzMTEiLz48L3N2Zz4=
```

5.3.7.4.4 Enable a Custom Application from the Workspace Catalog

Enterprise Gov Advanced

Enable a Custom Application from the Workspace Catalog

After creating a GitRepository, you can either use the DKP UI or the CLI to enable your custom applications. To deploy an application to selected clusters within a workspace, refer to the cluster-scoped configuration (see page 682) section.

> From within a workspace, you can enable applications to deploy. Verify that an application has successfully deployed via the CLI (see page 0).

5.3.7.4.4.1 Enable the Application Using the UI

Follow these steps:

1. From the top menu bar, select your target workspace.
2. Select **Applications** from the sidebar menu to browse the available applications from your configured repositories.
3. Select the three dot button from the bottom-right corner of the desired application tile, and then select **Enable**.
4. If available, select a version from the drop-down menu. This drop-down menu will only be visible if there is more than one version.

5. (Optional) If you want to override the default configuration values, copy your customized values into the text editor under **Configure Service** or upload your YAML file that contains the values:

```
someField: someValue
```

6. Confirm the details are correct, and then select the **Enable** button.

For all applications, you must provide a display name and an ID which is automatically generated based on what you enter for the display name, unless or until you edit the ID directly. The ID must be compliant with Kubernetes DNS subdomain name validation rules[534].

Alternately, you can use the CLI to enable your custom applications (see page 709).

5.3.7.4.4.2 Prerequisites

- Determine the name of the workspace where you wish to perform the deployments. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.
- Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace where the cluster is attached:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

5.3.7.4.4.3 Enable the Application using the CLI

Follow these steps:

1. Get the list of available applications to enable using the following command:

```
kubectl get apps -n ${WORKSPACE_NAMESPACE}
```

2. Deploy one of the supported applications from the list with an `AppDeployment` resource.
3. Within the `AppDeployment`, define the `appRef` to specify which `App` will be enabled:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: my-custom-app
  namespace: ${WORKSPACE_NAMESPACE}
spec:
```

534 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#dns-subdomain-names

```
  appRef:
    name: custom-app-0.0.1
    kind: App
EOF
```

- The `appRef.name` must match the app `name` from the list of available catalog applications.
- Create the resource in the workspace you just created, which instructs Kommander to deploy the `AppDeployment` to the `KommanderCluster`s in the same workspace.

5.3.7.4.4.4 Enable an Application with a Custom Configuration using the CLI

Follow these steps:

1. Provide the name of a `ConfigMap` in the `AppDeployment`, which provides custom configuration on top of the default configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: my-custom-app
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: custom-app-0.0.1
    kind: App
  configOverrides:
    name: my-custom-app-overrides
EOF
```

2. Create the `ConfigMap` with the name provided in the step above, with the custom configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: my-custom-app-overrides
data:
  values.yaml: |
```

```
    someField: someValue
EOF
```

Kommander waits for the `ConfigMap` to be present before deploying the `AppDeployment` to the managed or attached clusters in the Workspace.

5.3.7.4.4.5 Verify Applications

After completing the previous steps, your applications are enabled. Connect to the attached cluster and check the `HelmReleases` to verify the deployments:

```
kubectl get helmreleases -n ${WORKSPACE_NAMESPACE}
```

The output looks similar to this:

```
NAMESPACE                NAME            READY   STATUS
AGE
workspace-test-vjsfq     my-custom-app   True    Release reconciliation succeeded
7m3s
```

5.3.8 Workspace Role Bindings

Enterprise Gov Advanced

Workspace Role Bindings grant access to specified Workspace Roles for a specified group of people.

5.3.8.1 Prerequisites

- Before you can create a Workspace Role Binding, ensure you have created a workspace **Group**. A Group can contain one or several Identity Provider users, groups or both.

The syntax for the Identity Provider groups you add to a DKP Group varies depending on the context for which you have established an Identity Provider.

- If you have set up an identity provider globally, for **All Workspaces**:
  - For *groups*: Add an **Identity Provider Group** in the `oidc:<IdP_user_group>` format. For example, `oidc:engineering`.
  - For *users*: Add an **Identity Provider User** in the `<user_email>`. For example, `jane.doe@example.com`.

- If you have set up an identity provider for a **Specific Workspace**:
  - For *groups*: Add an **Identity Provider Group** in the `oidc:<workspace_name>:<IdP_user_group>` format. For example, `oidc:tenant-z:engineering`.
  - For *users*: Add an **Identity Provider User** in the `<workspace_ID>:<user_email>` format. For example, `tenant-z:jane.doe@example.com`.

    Run `kubectl get workspaces` to obtain a list of all existing workspaces. The `workspace_ID` is listed under the `NAME` column.

5.3.8.2 Configure Workspace Role Bindings

You can assign a role to this Kommander Group:

1. From the top menu bar, select your target workspace.
2. Select **Access Control** in the **Administration** section of the sidebar menu.
3. Select the **Cluster Role Bindings** tab, and then select the **Add Roles** button next to the group you want.
4. Select the Role, or Roles, you want from the drop-down menu and select **Save**.

It will take a few minutes for the resource to be created.

5.3.9 Multi-Tenancy in DKP

5.3.9.1 What Is Multi-Tenancy?

You can use workspaces to manage your tenants' environments separately, while still maintaining control over clusters and environments centrally. For example, if you operate as a **Managed Service Provider (MSP),** you can manage your clients clusters' lifecycles, resources, and applications. If you operate as an environment administrator, you can these resources per department, division, employee group, etc.

Here are some important concepts:

Multi-tenancy

Multi-tenancy in DKP is an architecture model where a single DKP Enterprise instance serves multiple organization's divisions, customers or tenants. In DKP, each tenant system is represented by a workspace. Each workspace and its resources can be isolated from other workspaces (by using separate Identity Providers), even though they all fall under a single Enterprise license.

Multi-tenant environments have at least two participating parties: the Enterprise license administrator (for example, an MSP), and one or several tenants.

Managed Service Provider or MSP

Managed Service Providers or MSPs are partner organizations that use DKP to facilitate cloud infrastructure services to their customers or tenants. For more information, see MSP Program (see page 105).

Tenant

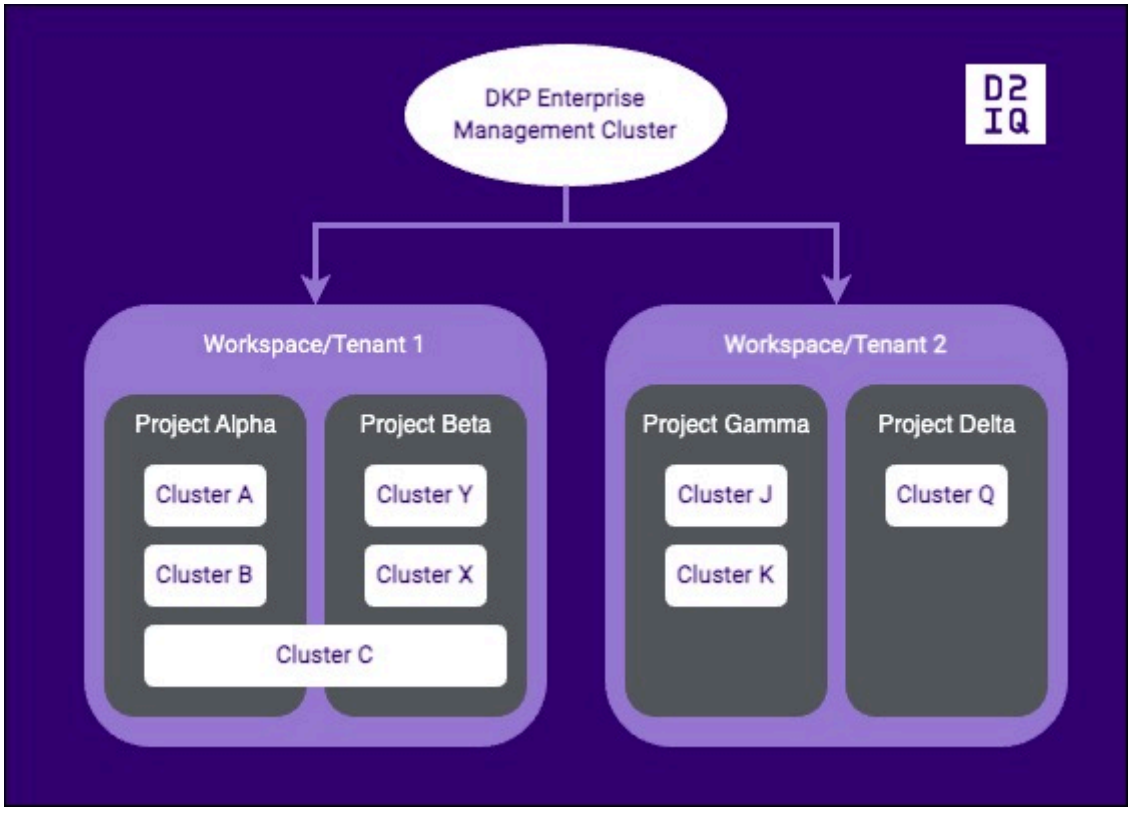
Tenants can be customers of Managed Service Provider partners. They outsource their cloud management requirements to MSPs, so they can focus on the development of their products.

Tenants can also be divisions within an organization that require a strict isolation from other divisions, for example, through differentiated access control.

In DKP, a workspace is assigned to a tenant.

5.3.9.2 How Does Access Control Work in Multi-Tenant Environments?

To isolate each tenant's information and environment, multi-tenancy allows you to configure an identity provider per workspace or tenant. In this setup, DKP keeps all workspaces and tenants separate and isolated from each other.



2 Multi-tenancy example with one Management cluster, and two tenants.

You, as a global administrator, manage tenant access at the Workspace level. A tenant can further adapt user access at the Project level.

Workspaces and Projects

Workspaces: In a multi-tenant system, workspaces and tenants are synonymous. You can set up an identity provider to control all workspaces, including the Management cluster's `kommander` workspace. You can then set up additional identity providers for each workspace/tenant, and generate a dedicated Login URL (see page 714), so each tenant has its own user access.

Projects: After you set up an identity provider per workspace/tenant, the tenant can choose to further narrow down access with an additional layer. A tenant can choose to organize clusters into projects and assign differentiated access to user groups with Project Role Bindings (see page 757).

By assigning clusters to one or several projects, you can enable more complex user access.

5.3.9.3 How Do I Enable Multi-Tenancy?

To enable multi-tenancy, you must:

- Configure an Identity Provider (see page 610) globally, if you want to use a single IdP to access all of your tenant's environments.
- Configure an Identity Provider (see page 610) per workspace. This way, each tenant has a dedicated IdP to access their workspace.
- Create DKP Identity Provider groups with the correct prefixes (see page 711) to map your existing IdP groups.
- Create a dedicated login URL for each tenant (see page 714). You can provide a workspace login link to each tenant for access to the DKP UI and for the generation of kubectl API access tokens (see page 619).

To enforce security, every tenant should be in a different AWS account, so they are truly independent of each other.

5.3.9.4 Next Step:

Generate a Dedicated URL Login for Each Tenant (see page 714)

5.3.9.5 Generate a Dedicated URL Login for Each Tenant

This page contains instructions on how to generate a workspace-specific URL to access the DKP UI.

By making this URL available to your tenant, you provide them with a dedicated login page, where users can enter their SSO credentials (see page 0) to access their workspace in the DKP UI and to where users can create a token to access a cluster's kubectl API (see page 619). Other tenants and their SSO configurations are not visible.

See Multi-Tenancy in DKP (see page 712) for more information around identity provider configuration (see page 714).

5.3.9.5.1 Prerequisites

- You have followed the steps in How do I enable multi-tenancy? (see page 714)
- You have Admin permissions and access to all workspaces.

5.3.9.5.2 Generate a Login URL

1. Set an environment variable to point at the workspace for which you want to generate a URL:
   Replace `<name_target_workspace>` with the workspace name. If you do not know the exact name of the workspace, run `kubectl get workspace` to get a list of all workspace names.

   ```
   export WORKSPACE_NAME=<name_target_workspace>
   ```

2. Generate a DKP UI login URL for that workspace:

   ```
   echo https://$(kubectl get kommandercluster -n kommander host-cluster -o
   jsonpath='{ .status.ingress.address }')/token/landing/${WORKSPACE_NAME}
   ```

   The output looks similar to this:

   ```
   https://example.com/token/landing/<WORKSPACE_NAME>
   ```

3. Share the output login URL with your tenant, so users can start accessing their workspace from the DKP UI.

The login page displays:

- Identity providers set globally
- Identity providers set for that specific workspace

The login page does not display any resources or workspaces for which the tenant has no permissions.

5.4 Projects

Enterprise Gov Advanced

Multi-cluster Configuration Management

Projects support the management of configMaps, continuous deployments, secrets, services, quotas, and role-based access control and multi-tenant logging by leveraging federated resources. When a Project is created, DKP creates a federated namespace that is propagated to the Kubernetes clusters associated with this Project.

Federation in this context means that a common configuration is pushed out from a central location (DKP) to all Kubernetes clusters, or a pre-defined subset group, under DKP management. This pre-defined subset group of Kubernetes clusters is called a Project.

Projects enable teams to deploy their configurations and services to clusters in a consistent way. Projects enable central IT or a business unit to share their Kubernetes clusters among several teams. Using Projects, DKP leverages Kubernetes Cluster Federation (KubeFed) to coordinate the configuration of multiple Kubernetes clusters.

Kommander allows a user to use labels to select, manually or dynamically, the Kubernetes clusters associated with a Project.

5.4.1 Project Namespace

Project Namespaces isolate configurations across clusters. Individual standard Kubernetes namespaces are automatically created on all clusters belonging to the project. When creating a new project, you can customize the Kubernetes namespace name that is created. It is the grouping of all of these individual standard Kubernetes namespaces that make up the concept of a Project Namespace. A Project Namespace is a Kommander specific concept.

5.4.2 Create a Project

When you create a Project, you must specify a Project Name, a Namespace Name (optional) and a way to allow Kommander to determine which Kubernetes clusters will be part of this project.

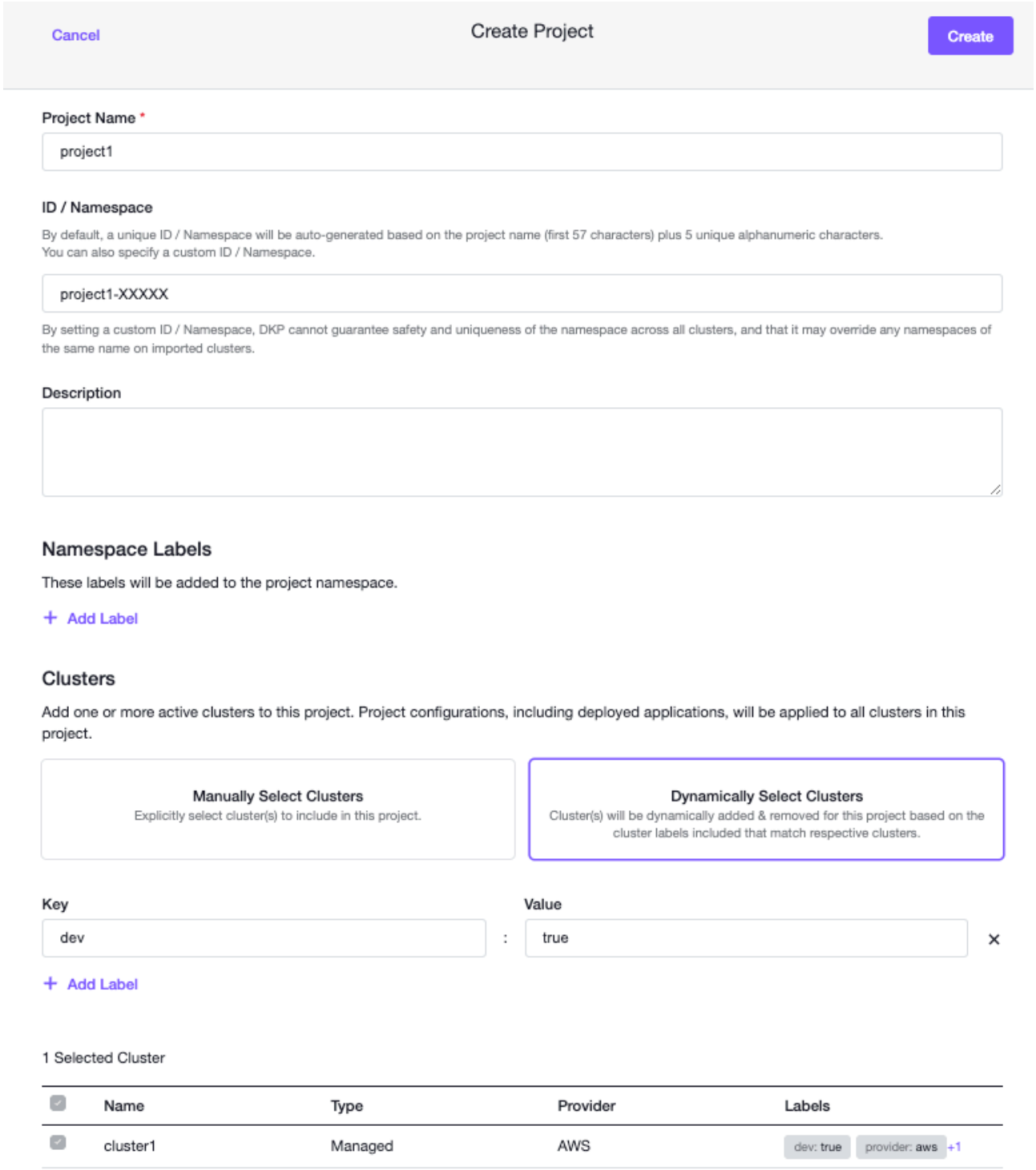
As mentioned previously, a Project Namespace corresponds to a Kubernetes Federated Namespace. By default, the name of the namespace is auto-generated based on the project name (first 57 characters) plus 5 unique alphanumeric characters. You can specify a namespace name, but you must ensure it does not conflict with any existing namespace on the target Kubernetes clusters, that will be a part of the Project.

To determine which Kubernetes clusters will be part of this project, you can either select manually existing clusters or define labels that Kommander will use to dynamically add clusters. The latter is recommended because it will allow you to deploy additional Kubernetes clusters later and to have them automatically associated with Projects based on their labels.

To create a Project, you can either use the DKP UI or create a Project object on the Kubernetes cluster where Kommander is running (using kubectl or the Kubernetes API). The latter allows you to configure Kommander resources in a declarative way. It is available for all kinds of Kommander resources.

5.4.3 Create a Project - UI Method

Here is an example of what it looks like to create a project using the DKP UI:



5.4.4 Create a Project - CLI Method

The following sample is a YAML Kubernetes object for creating a Kommander Project. This example does not work verbatim because it depends on a workspace name that has been previously created and does not exist by default in your cluster. Use this as an example format and fill in the workspace name and namespace name appropriately along with the proper labels.

```
apiVersion: workspaces.kommander.mesosphere.io/v1alpha1
kind: Project
metadata:
  name: My-Project-Name
  namespace: my-project-k8s-namespace-name
spec:
  workspaceRef:
    name: myworkspacename
  namespaceName: myworkspacename-di3tx
  placement:
    clusterSelector:
      matchLabels:
        cluster: prod
```

The following procedures are supported for projects:

- Project Applications (see page 718)
- Project Deployments (see page 740)
- Project Role Bindings (see page 757)
- Project Roles (see page 761)
- Project ConfigMaps (see page 763)
- Project Secrets (see page 765)
- Project Quotas & Limit Ranges (see page 766)
- Project Network Policies (see page 769)

5.4.5 Project Applications

Enterprise Gov Advanced

This section documents the applications and application types that you can utilize with DKP.

Application types are:

- Catalog Applications (see page 725) are either pre-packaged applications from the D2iQ Application Catalog, or custom applications that you maintain for your teams or organization.
    - DKP Applications (see page 726) are applications that are provided by D2iQ and added to the Catalog.
    - Custom Applications (see page 731) are applications you create and add to the Catalog.
- Platform Applications (see page 719) are applications integrated into Kommander.

When deploying and upgrading applications, platform applications come as a bundle; they are tested as a single unit and you must deploy or upgrade them in a single process, for each workspace. This means all clusters in a workspace have the same set and versions of platform applications deployed. Whereas catalog applications are individual, so you can deploy and upgrade them individually, for each project.

5.4.5.1 Project Platform Applications

Enterprise Gov Advanced

How project Platform applications work

The following table describes the list of applications that can be deployed to attached clusters within a project.

Review the Project Platform Application Requirements (see page 724) to ensure that the attached clusters in the project have sufficient resources.

From within a project, you can enable applications to deploy. Verify that an application has successfully deployed via the CLI (see page 667).

5.4.5.1.1 Enable Applications in a Project Using the UI

1. From the top menu bar, select your target workspace.
2. Select **Projects** from the sidebar menu.
3. Select your project from the list.
4. Select the **Applications** tab to browse the available applications.
5. Select the three dot button from the bottom-right corner of the desired application tile, and then select **Enable**.
6. To override the default configuration values, copy your values content into the text editor under **Configure Service** or just upload your yaml file that contains the values:

   ```
   someField: someValue
   ```

7. Select the **Enable** button.

To use the CLI to enable or disable applications, see Application Deployment (see page 667)

> ⚠ There may be dependencies between the applications, which are listed in Project Platform Application Dependencies (see page 723). Review them carefully prior to customizing to ensure that the applications are deployed successfully.

5.4.5.1.2 Platform Applications

NAME	APP ID	Deployed by default
project-grafana-logging-6.57.4	project-grafana-logging	False
project-grafana-loki-0.69.16	project-grafana-loki	False
project-logging-1.0.3	project-logging	False

5.4.5.1.3 Upgrade Platform Applications from the CLI

Platform Applications within a Project are automatically upgraded when the Workspace that a Project belongs to is upgraded. See Enterprise: Upgrade the Management Cluster and Platform Applications (see page 1719) for more information on how to upgrade these applications.

5.4.5.1.4 Deploy Project Platform Applications

Enterprise Gov Advanced

Deploy applications to attached clusters in a project using the CLI

This topic describes how to use the CLI to deploy an application to attached clusters within a project. To use the DKP UI to deploy applications, see Deploy Applications in a Project (see page 719).

See Project Platform Applications (see page 719) for a list of all applications and those that are enabled by default.

5.4.5.1.4.1 Prerequisites

Before you begin, you must have:

- A running cluster with Kommander installed (see page 127).
- An existing Kubernetes cluster attached to Kommander (see page 784).

Set the `WORKSPACE_NAME` environment variable to the name of the workspace where the cluster is attached:

```
export WORKSPACE_NAME=<workspace_name>
```

Set the `WORKSPACE_NAMESPACE` environment variable to the namespace of the above workspace:

```
export WORKSPACE_NAMESPACE=$(kubectl get namespace --
selector='workspaces.kommander.mesosphere.io/workspace-name=${WORKSPACE_NAME}' -o
jsonpath='{.items[0].metadata.name}')
```

Set the `PROJECT_NAME` environment variable to the name of the project in which the cluster is included:

```
export PROJECT_NAME=<project_name>
```

Set the `PROJECT_NAMESPACE` environment variable to the name of the above project's namespace:

```
export PROJECT_NAMESPACE=$(kubectl get project ${PROJECT_NAMESPACE} -n $
{WORKSPACE_NAMESPACE} -o jsonpath='{.status.namespaceRef.name}')
```

5.4.5.1.4.2 Deploy the Application

The list of available applications that can be deployed on the attached clusters in a project can be found in the project platform applications topic (see page 719).

1. Deploy one of the supported applications to your existing attached cluster (see page 784) with an `AppDeployment` resource. Provide the `appRef` and application version to specify which `App` is deployed:

   ```
   dkp create appdeployment project-grafana-logging --app project-grafana-
   logging-6.38.1 --workspace ${WORKSPACE_NAME} --project ${PROJECT_NAME}
   ```

2. Create the resource in the project you just created, which instructs Kommander to deploy the `AppDeployment` to the `KommanderClusters` in the same project.

- The `appRef.name` must match the app `name` from the list of available applications.
- Observe that the `dkp create` command must be run with both the `--workspace` and `--project` flags for project platform applications.

5.4.5.1.4.3 Deploy an Application with a Custom Configuration

Follow these steps:

1. Create the `AppDeployment` and provide the name of a `ConfigMap`, which provides custom configuration on top of the default configuration:

```
dkp create appdeployment project-grafana-logging --app project-grafana-
logging-6.38.1 --config-overrides project-grafana-logging-overrides --workspace
${WORKSPACE_NAME} --project ${PROJECT_NAMESPACE}
```

2. Create the `ConfigMap` with the name provided in the step above, with the custom configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${PROJECT_NAMESPACE}
  name: project-grafana-logging-overrides
data:
  values.yaml: |
    datasources:
      datasources.yaml:
        apiVersion: 1
        datasources:
        - name: Loki
          type: loki
          url: "http://project-grafana-loki-loki-distributed-gateway"
          access: proxy
          isDefault: false
EOF
```

Kommander waits for the `ConfigMap` to be present before deploying the `AppDeployment` to the attached clusters.

5.4.5.1.4.4 Verify Applications

The applications are now enabled.

Export the `project_namespace` with this command:

```
export PROJECT_NAMESPACE=<project_namespace>
```

Connect to the attached cluster and check the `HelmReleases` to verify the deployment:

```
kubectl get helmreleases -n ${PROJECT_NAMESPACE}
NAMESPACE               NAME                          READY   STATUS
AGE
project-test-vjsfq      project-grafana-logging       True    Release reconciliation
succeeded   7m3s
```

Some of the supported applications have dependencies on other applications. See Project Application Dependencies (see page 723) for that table.

5.4.5.1.5 Project Platform Application Dependencies

Enterprise Gov Advanced

Dependencies between project platform applications

There are many dependencies between the applications that are deployed to a project's attached clusters. It is important to note these dependencies when customizing the platform applications to ensure that your services are properly deployed to the clusters. For more information on how to customize platform applications, see Platform Application Deployment (see page 720).

5.4.5.1.5.1 Application Dependencies

When deploying or troubleshooting applications, it helps to understand how applications interact and may require other applications as dependencies.

If an application's dependency does not successfully deploy, the application requiring that dependency does not successfully deploy.

The following sections detail information about the platform applications.

5.4.5.1.5.2 Logging

Collects logs over time from Kubernetes pods deployed in the project namespace. Also provides the ability to visualize and query the aggregated logs.

- project-logging[535]: Defines resources for the Logging Operator which uses them to direct the project's logs to its respective Grafana Loki application.
- project-grafana-loki[536]: A horizontally-scalable, highly-available, multi-tenant log aggregation system inspired by Prometheus.
- project-grafana-logging[537]: Logging dashboard used to view logs aggregated to Grafana Loki.

535 https://grafana.com/oss/grafana/
536 https://grafana.com/oss/loki/
537 https://grafana.com/oss/grafana/

The project logging applications depend on the workspace logging applications (see page 931) being deployed.

Application	Required Dependencies
project-logging	logging-operator (workspace)
project-grafana-loki	project-logging, grafana-loki (workspace), logging-operator (workspace)
project-grafana-logging	project-grafana-loki

5.4.5.1.6 Project Platform Application Configuration Requirements

Enterprise Gov Advanced

5.4.5.1.6.1 Project Platform Application Descriptions and Resource Requirements

Platform applications require more resources than solely deploying or attaching clusters into a project. Your cluster must have sufficient resources when deploying or attaching to ensure that the applications are installed successfully.

The following table describes all the platform applications that are available to the clusters in a project, minimum resource and persistent storage requirements, and whether they are enabled by default.

Name	Minimum Resources Suggested	Minimum Persistent Storage Required	Deployed by Default	Default Priority Class
project-grafana-logging	cpu: 200m memory: 100Mi		No	DKP Critical (100002000)
project-grafana-loki		# of PVs: 3 PV sizes: 10Gi x 3 (total: 30Gi)	No	DKP Critical (100002000)
project-logging			No	DKP Critical (100002000)

5.4.5.2 Project Catalog Applications

Enterprise Gov Advanced

Catalog applications are any third-party or open source applications that appear in the Catalog. These can be DKP applications provided by D2iQ for use in your environment, or Custom Applications (see page 731) that can be used but are not supported by D2iQ.

- Project Catalog Applications Upgrades (see page 725)
- Project-level DKP Applications (see page 726)
- Use Custom Resources with Workspace Catalog Applications (see page 731)
- Custom Project Applications (see page 731)

5.4.5.2.1 Project Catalog Applications Upgrades

Enterprise Gov Advanced

Before upgrading your catalog applications, verify the current and supported versions of the application. Also, keep in mind the distinction between Platform applications and Catalog applications. Platform applications are deployed and upgraded as a set for each cluster or workspace. Catalog applications are deployed separately, so that you can deploy and upgrade them individually for each project.

> Catalog applications must be upgraded to the latest version BEFORE upgrading the Konvoy component for Managed clusters or Kubernetes version for attached clusters.

5.4.5.2.1.1 Upgrade with the UI

Follow these steps to upgrade an application from the DKP UI:

1. From the top menu bar, select your target workspace.
2. From the side menu bar, select **Projects**.
3. Select your target project.
4. Select **Applications** from the project menu bar.
5. Select the three dot button from the bottom-right corner of the desired application tile, and then select **Edit**.
6. Select the **Version** drop-down, and select a new version. This drop-down will only be available if there is a newer version to upgrade to.
7. Select **Save**.

5.4.5.2.1.2 Upgrade with the CLI

1. To see what app(s) and app versions are available to upgrade, run the following command:
   NOTE: The `APP ID` column displays the available apps and the versions available to upgrade.

   ```
   kubectl get apps -n ${PROJECT_NAMESPACE}
   ```

2. Run the following command to upgrade an application from the DKP CLI:

   ```
   dkp upgrade catalogapp <appdeployment-name> --workspace=my-workspace --
   project=my-project --to-version=<version.number>
   ```

As an example, the following command upgrades the Kafka Operator application, named `kafka-operator-abc`, in a workspace to version `0.25.1`:

```
dkp upgrade catalogapp kafka-operator-abc --workspace=my-workspace --to-version=0.25.1
```

Platform applications cannot be upgraded on a one-off basis, and must be upgraded in a single process for each workspace. If you attempt to upgrade a platform application with these commands, you receive an error and the application is not upgraded.

5.4.5.2.2 Project-level DKP Applications

Enterprise Gov Advanced

DKP applications are catalog applications provided by D2iQ for use in your environment.

Some DKP workspace catalog applications will provision `CustomResourceDefinitions`, which allow you to deploy Custom Resources to a Project. See your DKP workspace catalog application's documentation for instructions.

5.4.5.2.2.1 Zookeeper in a Project

Enterprise Gov Advanced

Deploying ZooKeeper in a project

Get started

To get started with creating ZooKeeper clusters in your project namespace, you first need to deploy the ZooKeeper operator (see page 695) in the workspace where the project exists.

After you deploy the ZooKeeper operator, you can create ZooKeeper Clusters by applying a `ZookeeperCluster` custom resource **on each attached cluster** in a project's namespace.

A Helm chart[538] exists in the ZooKeeper operator repository that can assist with deploying ZooKeeper clusters.

Example Deployment

> If you need to manage these custom resources across all clusters in a project, it is recommended you use Project Deployments (see page 740) which enables you to leverage GitOps to deploy the resources. Otherwise, you will need to create the resources manually in each cluster.

Follow these steps to deploy a ZooKeeper cluster in a project namespace. This procedure results in a running ZooKeeper cluster, ready for use in your project's namespace.

1. Set the `PROJECT_NAMESPACE` environment variable to the name of your project's namespace:

   ```
   export PROJECT_NAMESPACE=<project namespace>
   ```

2. Create a ZooKeeper Cluster custom resource in your project namespace

   ```
   kubectl apply -f - <<EOF
   apiVersion: zookeeper.pravega.io/v1beta1
   kind: ZookeeperCluster
   metadata:
     name: zookeeper
     namespace: ${PROJECT_NAMESPACE}
   spec:
     replicas: 1
   EOF
   ```

3. Check the status of your ZooKeeper cluster using `kubectl`:

   ```
   kubectl get zookeeperclusters -n ${PROJECT_NAMESPACE}
   ```

538 https://github.com/pravega/zookeeper-operator/tree/master/charts/zookeeper

```
NAME        REPLICAS   READY REPLICAS    VERSION   DESIRED VERSION   INTERNAL
ENDPOINT    EXTERNAL ENDPOINT   AGE
zookeeper   1          1                0.2.15    0.2.15            10.100.200
.18:2181   N/A                94s
```

Delete ZooKeeper clusters

Follow these steps to delete the Zookeeper clusters.

1. View `ZookeeperClusters` in all namespaces:

   ```
   kubectl get zookeeperclusters -A
   ```

2. Delete a specific `ZookeeperCluster`:

   ```
   kubectl -n ${PROJECT_NAMESPACE} delete zookeepercluster <name of
   zookeepercluster>
   ```

Resources

- ZooKeeper Operator Documentation[539]
- ZooKeeper Cluster Helm Chart[540]
- ZooKeeper Documentation[541]

5.4.5.2.2.2 Kafka in a Project

Enterprise Gov Advanced

Deploying Kafka in a project

Prerequisites

To get started with creating and managing a Kafka Cluster in a project, you must:

- Deploy the Kafka operator (see page 693) in the workspace where the project exists.
- Deploy the ZooKeeper operator (see page 695) in the workspace where the project exists.
- Deploy Zookeeper in a Project (see page 726) in the same project where you want to enable Kafka.

Get Started

539 https://github.com/pravega/zookeeper-operator
540 https://github.com/pravega/zookeeper-operator/tree/master/charts/zookeeper
541 https://zookeeper.apache.org/documentation

After deploying the Kafka operator, create Kafka Clusters by applying a `KafkaCluster` custom resource **on each attached cluster** in a project's namespace. Refer to the Kafka operator repository[542] for examples of the custom resources and their configurations.

Example Deployment

If you need to manage these custom resources across all clusters in a project, it is recommended you use Project Deployments (see page 740) which enables you to leverage GitOps to deploy the resources. Otherwise, you will need to create the custom resources manually in each cluster.

This example deployment walks you through deploying a Kafka cluster in a project namespace. The result of this procedure is a running ZooKeeper cluster and Kafka cluster ready for use in your project's namespace.

1. Ensure you deployed Zookeeper in a Project (see page 726).
2. Set the `PROJECT_NAMESPACE` environment variable to the name of your project's namespace:

   ```
   export PROJECT_NAMESPACE=<project namespace>
   ```

3. Obtain the Kafka Operator version you deployed in the workspace:
   Replace `<target_namespace>` with the namespace where you deployed Kafka:

   ```
   kubectl get appdeployments.apps.kommander.d2iq.io -n <target_namespace> -o
   template="{{ .spec.appRef.name }}" kafka-operator
   ```

   The output prints the Kafka Operator version.
4. Use the Kafka Operator version to download the `simplekafkacluster.yaml` file you require:

In the following URL, replace `/v0.25.1/` with the Kafka version you obtained in the previous step and download the file.
https://raw.githubusercontent.com/banzaicloud/koperator/v0.25.1/config/samples/simplekafkacluster.yaml
In order to use a CVE-free kafka image, set `clusterImage` value to `ghcr.io/banzaicloud/kafka:2.13-3.4.1` (similarly to workspace installation in https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919406/Kafka+in+a+Workspace#Using-a-custom-image-for-a-zookeeper-cluster (see page 0)).

542 https://github.com/banzaicloud/koperator/tree/master/config/samples

5. Open and edit the downloaded file to use the correct Zookeeper Cluster address:
   Replace `<project_namespace>` with the target project namespace.

```
zkAddresses:
    - "zookeeper-client.<project_namespace>:2181"
```

6. Apply the `KafkaCluster` configuration to your project's namespace:

```
kubectl apply -n ${PROJECT_NAMESPACE} -f simplekafkacluster.yaml
```

7. Check the status of your Kafka cluster using `kubectl`:

```
kubectl -n ${PROJECT_NAMESPACE} get kafkaclusters
```

   The output should look similar to this:

```
NAME    CLUSTER STATE    CLUSTER ALERT COUNT   LAST SUCCESSFUL UPGRADE
UPGRADE ERROR COUNT   AGE
kafka   ClusterRunning   0                                           0
79m
```

With both the ZooKeeper cluster and Kafka cluster running in your project's namespace, refer to the Kafka Operator documentation[543] for information on how to test and verify they are working as expected in. When performing those steps, ensure you substitute: `zookeeper-client.<project namespace>:2181` anywhere that the zookeeper client address is mentioned.

Delete Kafka Custom Resources

Follow these steps to delete the Kafka custom resources.

1. View all Kafka resources in the cluster:

```
kubectl get kafkaclusters -A
kubectl get kafkausers -A
Kubectl get kafkatopics -A
```

2. Delete a `KafkaCluster` example:

```
kubectl -n ${PROJECT_NAMESPACE} delete kafkacluster <name of KafkaCluster>
```

543 https://banzaicloud.com/docs/supertubes/kafka-operator/test/

Resources

- Kafka Operator Documentation[544]
- Kafka Operator GitHub Repository[545]
- Sample Kafka Operator Custom Resources[546]
- Apache Kafka Documentation[547]

5.4.5.2.3 Use Custom Resources with Workspace Catalog Applications

Enterprise Gov Advanced

Some workspace catalog applications provision some `CustomResourceDefinition`, which allow you to deploy Custom Resources. Refer to your workspace catalog application's documentation for instructions.

5.4.5.2.4 Custom Project Applications

Enterprise Gov Advanced

Custom applications are third-party applications you have added to the Kommander Catalog.

Custom applications are any third-party applications that are not provided in the DKP Application Catalog. Custom applications can leverage applications from the DKP Catalog or be fully-customized. There is no expectation of support by D2iQ for a Custom application. Custom applications can be deployed on Konvoy clusters or on any D2iQ supported 3rd party Kubernetes distribution.

- Projects - Create a Git Repository (see page 731)
- Projects - Git repository structure (see page 733)
- Projects - Application Metadata (see page 735)
- Enable a Custom Application from the Project Catalog (see page 737)

5.4.5.2.4.1 Projects - Create a Git Repository

Enterprise Gov Advanced

Create a Git Repository in the Project namespace

Use the CLI to create the GitRepository resource and add a new repository to your Project.

1. Refer to air-gapped setup instructions (see page 1533), if you are running in air-gapped environment.
2. Set the `PROJECT_NAMESPACE` environment variable to the name of your project's namespace:

544 https://banzaicloud.com/docs/supertubes/kafka-operator/
545 https://github.com/banzaicloud/koperator
546 https://github.com/banzaicloud/koperator/tree/master/config/samples
547 https://kafka.apache.org/documentation/

```
export PROJECT_NAMESPACE=<project_namespace>
```

3. Adapt the URL of your Git repository.

```
kubectl apply -f - <<EOF
apiVersion: source.toolkit.fluxcd.io/v1
kind: GitRepository
metadata:
  name: example-repo
  namespace: ${PROJECT_NAMESPACE}
spec:
  interval: 1m0s
  ref:
    branch: <your-target-branch-name> # e.g., main
  timeout: 20s
  url: https://github.com/<example-org>/<example-repo>
EOF
```

4. Ensure the status of the `GitRepository` signals a ready state:

```
kubectl get gitrepository example-repo -n ${PROJECT_NAMESPACE}
```

The repository commit also displays the ready state:

```
NAME         URL                                                     READY
STATUS                                                          AGE
example-repo https://github.com/example-org/example-repo             True
Fetched revision: master/6c54bd1722604bd03d25dcac7a31c44ff4e03c6a   11m
```

For more information on the GitRepository resource fields and how to make Flux aware of credentials required to access a private Git repository, see the Flux documentation[548].

Troubleshoot

To troubleshoot issues with adding the GitRepository, review the following logs:

```
kubectl -n kommander-flux logs -l app=source-controller
[...]
kubectl -n kommander-flux logs -l app=kustomize-controller
[...]
kubectl -n kommander-flux logs -l app=helm-controller
[...]
```

548 https://fluxcd.io/docs/components/source/gitrepositories/

Related Information

- Flux[549]
- Flux docs[550]

5.4.5.2.4.2 Projects - Git repository structure

Enterprise Gov Advanced

Git repositories must be structured in a specific manner for defined applications to be processed by Kommander.

You must structure your git repository based on the following guidelines, for your applications to be processed properly by Kommander so that they can be deployed.

Git Repository Directory Structure

Use the following basic directory structure for your git repository:

```
├── helm-repositories
│   ├── <helm repository 1>
│   │   ├── kustomization.yaml
│   │   └── <helm repository name>.yaml
│   └── <helm repository 2>
│       ├── kustomization.yaml
│       └── <helm repository name>.yaml
└── services
    ├── <app name>
    │   ├── <app version1> # semantic version of the app helm chart. e.g., 1.2.3
    │   │   ├── defaults
    │   │   │   ├── cm.yaml
    │   │   │   └── kustomization.yaml
    │   │   ├── <app name>.yaml
    │   │   └── kustomization.yaml
    │   ├── <app version2> # another semantic version of the app helm chart. e.g.,
2.3.4
    │   │   ├── defaults
    │   │   │   ├── cm.yaml
    │   │   │   └── kustomization.yaml
    │   │   ├── <app name>.yaml
    │   │   └── kustomization.yaml
    │   └── metadata.yaml
    └── <another app name>
    ...
```

549 https://fluxcd.io/
550 https://fluxcd.io/docs

- Define applications in the `services/` directory.
- You can define multiple versions of an application, under different directories nested under the `services/<app name>/` directory.
- Define application manifests, such as a HelmRelease[551], under each versioned directory `services/<app name>/<version>/` in `<app name>.yaml` which is listed in the `kustomization.yaml` Kubernetes Kustomization file. For more information, see the Kubernetes Kustomization docs[552].
- Define the default values ConfigMap for `HelmReleases` in the `services/<app name>/<version>/defaults` directory, accompanied by a `kustomization.yaml` Kubernetes Kustomization file pointing to the `ConfigMap` file.
- Define the `metadata.yaml` of each application under the `services/<app name>/` directory. For more information, see the Application Metadata docs (see page 706).

See the DKP Catalog repository[553] for an example of how to structure custom catalog Git repositories.

Helm Repositories

You must include the `HelmRepository` that is referenced in each `HelmRelease`'s `Chart` spec.

Each `services/<app name>/<version>/kustomization.yaml` must include the path of the YAML file that defines the `HelmRepository`. For example:

```
# services/<app name>/<version>/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - <app name>.yaml
  - ../../../helm-repositories/<helm repository 1>
```

For more information, see the flux documentation about HelmRepositories[554].

Substitution Variables

Some substitution variables[555] are provided.

- `${releaseName}`: For each App deployment, this variable is set to the `AppDeployment` name. Use this variable to prefix the names of any resources that are defined in the application directory in the Git repository so that multiple instances of the same application can be deployed. If you create resources without using the `releaseName` prefix (or suffix) in the name field, there can be conflicts if the same named resource is created in that same namespace.

551 https://fluxcd.io/docs/components/helm/helmreleases/
552 https://kubectl.docs.kubernetes.io/references/kustomize/kustomization/
553 https://github.com/mesosphere/dkp-catalog-applications
554 https://fluxcd.io/docs/components/source/helmrepositories/
555 https://fluxcd.io/docs/components/kustomize/kustomization/#variable-substitution

- `${releaseNamespace}` : The namespace of the Project.
- `${workspaceNamespace}` : The namespace of the Workspace that the Project belongs to.

Related Information

- Flux[556]
- Flux docs[557]
- Getting started with Flux guide[558]

5.4.5.2.4.3 Projects - Application Metadata

Enterprise Gov Advanced

To display more information about custom applications in the UI, define a metadata.yaml file for each application in the git repository.

You can define how custom applications display in the DKP UI by defining a `metadata.yaml` file for each application in your git repository. You must define this file at `services/<application>/metadata.yaml` for it to process correctly.

You can define the following fields:

Field	Default	Description
displayName	falls back to App ID	Display name of the application for the UI.
description	“”	Short description, should be a sentence or two, displayed in the UI on the application card.
category	general	1 or more categories for this application. Categories are used to group applications in the UI.
overview		Markdown overview used on the application detail page in the UI.

556 https://fluxcd.io/
557 https://fluxcd.io/docs
558 https://fluxcd.io/docs/get-started/

Field	Default	Description
icon		Base64 encoded icon SVG file used for application logos in the UI.
scope	project	List of scopes, can be set only to project or workspace currently.

None of these fields are required for the application to display in the UI.

Here is an example `metadata.yaml` file:

```
displayName: Prometheus Monitoring Stack
description: Stack of applications that collect metrics and provides visualization
and alerting capabilities. Includes Prometheus, Prometheus Alertmanager and Grafana.
category:
  - monitoring
overview: >
  # Overview
  A stack of applications that collects metrics and provides visualization and
alerting capabilities. Includes Prometheus, Prometheus Alertmanager and Grafana.

  ## Dashboards
  By deploying the Prometheus Monitoring Stack, the following platform applications
and their respective dashboards are deployed. After deployment to clusters in a
workspace, the dashboards are available to access from a respective cluster's detail
page.

  ### Prometheus

  A software application for event monitoring and alerting. It records real-time
metrics in a time series database built using a HTTP pull model, with flexible and
real-time alerting.

  - [Prometheus Documentation - Overview](https://prometheus.io/docs/introduction/
overview/)

  ### Prometheus Alertmanager
  A Prometheus component that enables you to configure and manage alerts sent by the
Prometheus server and to route them to notification, paging, and automation systems.

  - [Prometheus Alertmanager Documentation - Overview](https://prometheus.io/docs/
alerting/latest/alertmanager/)

  ### Grafana
  A monitoring dashboard from Grafana that can be used to visualize metrics collected
by Prometheus.
```

```
  - [Grafana Documentation](https://grafana.com/docs/)
icon:
```

```
PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHZpZXdCb3g9IjAgMCAzMDAgMzAwIiBzd
HlsZT0iZW5hYmxlLWJhY2tncm91bmQ6bmV3IDAgMCAzMDAgMzAwIiB4bWw6c3BhY2U9InByZXNlcnZlIj48cG
F0aCBkPSJNMTUwIDUwQzk0LjggNTAgNTAgOTQuOCA1MCAxNTBzNDQuOCAxMDAgMTAwIDEwMCAxMDAtNDQuOCA
xMDAtMTAwUzIwNS4yIDUwIDE1MCA1MHptMCAxODcuMmMtMTUuNyAwLTI4LjUtMTAuNS0yOC41LTIzLjRoNTYu
OWMuMSAxMi45LTEyLjcgMjMuNC0yOC40IDIzLjR6bTQ3LTMxLjJoLTk0di0xN2g5NHYxN3ptLS4zLTI1LjloL
TkzLjRjLS4zLS40LS42LS43LS45LTEuMS05LjYtMTEuNy0xMS45LTE3LjgtMTQuMS0yNCAwLS4yIDExLjcgMi
40IDIwIDQuMyAwIDAgNC4zIDEgMTAuNSAyLjEtNi03LTkuNi0xNi05LjYtMjUuMSAwLTIwIDE1LjQtMzcuNiA
5LjgtNTEuNyA1LjQuNCAxMS4yIDExLjQgMTEuNiAyOC41IDUuNy03LjkgOC4xLTIyLjQgOC4xLTMxLjMgMC05
LjIgNi4xLTE5LjkgMTIuMS0yMC4yLTUuNCA4LjkgMS40IDE2LjUgNy40IDM1LjUgMi4zIDcuMSAyIDE5LjEgM
y43IDI2LjcuNi0xNS44IDMuMy0zOC44IDEzLjMtNDYuNy00LjQgMTAgLjcgMjIuNSA0LjEgMjguNSA1LjYgOS
43IDkgMTcuMSA5IDMxIDAgOS4zLTMuNCAxOC4xLTkuMyAyNSA2LjYtMS4yIDExLjItMi40IDExLjItMi40bDI
xLjQtNC4yYy4xIDAtMyAxMi44LTE0LjkgMjUuMXoiIHN0eWxlPSJmaWxsOiNmODQzMTEiLz48L3N2Zz4=
```


5.4.5.2.4.4 Enable a Custom Application from the Project Catalog

Enterprise Gov Advanced

Enable a Custom Application from the Project Catalog

After creating a GitRepository, you can either use the DKP UI or the CLI to enable your custom applications.

> From within a project, you can enable applications to deploy. Verify that an application has successfully deployed via the CLI.

Enable the Application using the UI

1. From the top menu bar, select your target workspace.
2. Select **Projects** from the sidebar menu.
3. Select your project from the list.
4. Select the **Applications** tab to browse the available applications from your configured repositories.
5. Select the three dot button from the bottom-right corner of the desired application tile, and then select **Enable**.
6. If available, select a version from the drop-down menu. This drop-down menu will only be visible if there is more than one version.
7. (Optional) If you want to override the default configuration values, copy your values content into the text editor under **Configure Service** or just upload your YAML file that contains the values:

   ```
   someField: someValue
   ```

8. Confirm the details are correct, and then select the **Enable** button.

For all applications, you must provide a display name and an ID which is automatically generated based on what you enter for the display name, unless or until you edit the ID directly. The ID must be compliant with Kubernetes DNS subdomain name validation rules[559].

Alternately, you can use the CLI (see page 0) to enable your custom applications in the section below.

Enable the Application using the CLI

Follow these steps:

1. Set the `PROJECT_NAMESPACE` environment variable to the name of the project's namespace:

```
export PROJECT_NAMESPACE=<project_namespace>
```

2. Get the list of available applications to enable using the following command:

```
kubectl get apps -n ${PROJECT_NAMESPACE}
```

3. Enable one of the supported applications from the list with an `AppDeployment` resource.
4. Within the `AppDeployment` resource. Provide the `appRef` and application version to specify which `App` is deployed:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: my-custom-app
  namespace: ${PROJECT_NAMESPACE}
spec:
  appRef:
    name: custom-app-0.0.1
    kind: App
EOF
```

The `appRef.name` must match the app `name` from the list of available applications.

Enable an Application with a Custom Configuration using the CLI

559 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#dns-subdomain-names

Follow these steps:

1. Provide the name of a `ConfigMap` in the `AppDeployment`, which provides custom configuration on top of the default configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: my-custom-app
  namespace: ${PROJECT_NAMESPACE}
spec:
  appRef:
    name: custom-app-0.0.1
    kind: App
  configOverrides:
    name: my-custom-app-overrides
EOF
```

2. Create the `ConfigMap` with the name provided in the step above, with the custom configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${PROJECT_NAMESPACE}
  name: my-custom-app-overrides
data:
  values.yaml: |
    someField: someValue
EOF
```

Kommander waits for the `ConfigMap` to be present before enabling the `AppDeployment` to the attached clusters in the Project.

Verify Applications

After completing the previous steps, your applications are enabled. Connect to the attached cluster and check the `HelmReleases` to verify the deployments:

```
kubectl get helmreleases -n ${PROJECT_NAMESPACE}
```

The output looks similar to this:

```
NAMESPACE               NAME            READY   STATUS
AGE
project-test-vjsfq      my-custom-app   True    Release reconciliation succeeded
7m3s
```

Upgrade Custom Applications

You must maintain your custom applications manually. When upgrading DKP, ensure you validate for compatibility issues any custom applications you run against the current version of Kubernetes. We recommend upgrading to the latest compatible application versions as soon as possible.

5.4.5.3 Project AppDeployments

Enterprise Gov Advanced

An `AppDeployment` is a Custom Resource created by DKP with the purpose of deploying applications (platform, DKP catalog and custom applications). For more information about these Custom Resources and how to customize them, refer to the AppDeployments (see page 644) documentation.

5.4.6 Project Deployments

Enterprise Gov Advanced

Use Project Deployments to manage GitOps based Continuous Deployments

You can configure Kommander Projects with GitOps-based Continuous Deployments for federation of your Applications to associated clusters of the project. This is backed by Flux, which enables software and applications to be continuously deployed (CD) using GitOps processes. GitOps enables the application to be deployed as per a manifest that is stored in a Git repository. This ensures that the application deployment can be automated, audited, and declaratively deployed to the infrastructure.

- What is GitOps? (see page 740)
- Continuous Delivery with GitOps (see page 741)
- Continuous Deployment (see page 750)
- Project Deployments Troubleshooting (see page 754)
- View Helm Releases (see page 754)

5.4.6.1 What is GitOps?

GitOps is a modern software deployment strategy. The configuration that describes how your application is deployed to a cluster are stored in a Git repository. The configuration is continuously synchronized from the Git repository to the cluster, ensuring that the specified state of the cluster always matches what is defined in the “GitOps” Git repository.

The benefits of using a GitOps deployment strategy are:

- Familiar, collaborative change and review process. Engineers are intimately familiar with Git-based workflows: branches, pull requests, code reviews, etc. GitOps leverages this experience to control the deployment of software and updates to catch issues early.
- Clear change log and audit trail. The Git commit log serves as an audit trail to answer the question: “who changed what, and when?” Having such information available, you can contact the right people when fixing or prioritizing a production incident to determine the why and correctly resolve the issue as quickly as possible. Additionally, Kommander’s CD component (Flux CD) maintains a separate audit trail in the form of Kubernetes Events, as changes to a Git repository don’t include exactly when those changes were deployed.
- Avoid configuration drift. The scope of manual changes made by operators expands over time. It soon becomes difficult to know which cluster configuration is critical and which is left over from temporary workarounds or live debugging. Over time, changing a project configuration or replicating a deployment to a new environment becomes a daunting task. GitOps supports simple, reproducible deployment to multiple different clusters by having a single source of truth for cluster and application configuration.

That said, there are some cases when live debugging is necessary in order to resolve an incident in the minimum amount of time. In such cases, pull-request-based workflow adds precious time to resolution for critical production outages. Kommander’s CD strategy supports this scenario by letting you disable the auto sync feature. After auto sync is disabled, Flux will stop synchronizing the cluster state from the GitOps git repository. This lets you use `kubectl`, `helm`, or whichever tool you need to resolve the issue.

5.4.6.2 Continuous Delivery with GitOps

DKP enables software and applications to be continuously delivered (CD) using GitOps processes. GitOps enables you to deploy an application according to a manifest that is stored in a Git repository. This ensures that the application deployment can be automated, audited, and declaratively deployed to the infrastructure.

This section contains step-by-step tutorials for performing some common deployment-related tasks using DKP. All tutorials begin with a Prerequisites section that contains links to any steps that need to be taken first. This means you can visit any tutorial to get started.

- Store secrets in GitOps repository using SealedSecrets (see page 741)
- Deploy a Sample App from DKP GitOps (see page 745)

5.4.6.2.1 Store secrets in GitOps repository using SealedSecrets

5.4.6.2.1.1 Securely managing secrets in a GitOps workflow using SealedSecrets

For security reasons, Kubernetes secrets are usually the only resource that cannot be managed with a GitOps workflow. Instead of managing secrets outside of GitOps and having to use a third-party tool like Vault, SealedSecrets provides a way to keep all the advantages of using a GitOps workflow while avoiding exposing secrets. SealedSecrets is composed of two main components:

- A CLI (Kubeseal) to encrypt secrets.
- A cluster-side controller to decrypt the sealed secrets into regular Kubernetes secrets. Only this controller can decrypt sealed secrets, not even the original author.

This tutorial describes how to install these two components, configure the controller, and add or remove sealed secrets.

5.4.6.2.1.2 Set up

These instructions are used as an example. For instructions on the latest release, see the release page[560]. For full documentation on SealedSecrets, see the GitHub repo[561].

5.4.6.2.1.3 Install Kubeseal CLI to Encrypt Your Secrets

- On MacOS

```
brew install kubeseal
```

- On Linux

```
wget https://github.com/bitnami-labs/sealed-secrets/releases/download/v0.18.1/
kubeseal-0.18.1-linux-amd64.tar.gz -O kubeseal
sudo install -m 755 kubeseal /usr/local/bin/kubeseal
```

5.4.6.2.1.4 Install the SealedSecrets Controller on Your Cluster

This controller will be able to decrypt SealedSecrets and create Kubernetes secrets.

1. Create the controller:

```
kubectl apply -f https://github.com/bitnami-labs/sealed-secrets/releases/
download/v0.18.1/controller.yaml
```

2. Fetch the certificate that you will use to encrypt your secrets into sealed secrets:

```
kubeseal --fetch-cert > mycert.pem
```

3. Commit `mycert.pem` to your git repo.

5.4.6.2.1.5 Add a Secret

Secrets can be securely added to Git using sealed secrets:

1. Export the project namespace that you are using for your GitOps repository

560 https://github.com/bitnami-labs/sealed-secrets/releases
561 https://github.com/bitnami-labs/sealed-secrets

```
export PROJECT_NAMESPACE=<your-project-with-gitops>
```

2. Create a Kubernetes secret and pipe it into kubeseal[562] using the certificate `mycert.pem` that you fetched from the controller in the setup:

```
echo '---' >> secrets.yaml
kubectl create secret -n ${PROJECT_NAMESPACE} generic mysecret --dry-run=client
-o yaml --from-literal=my-secret=value | \
    kubeseal --format yaml --cert mycert.pem >> secrets.yaml
```

3. Go to the end of `secrets.yaml` where you just added your new sealed secret. Remove any "creationTimestamp" fields from the YAML.
4. Apply the `secrets.yaml` file to your namespace. If you do not have permission, commit your changes to the repo and let FluxCD apply the changes for you.

```
kubectl apply -f secrets.yaml
```

5. The sealed secret controller will then decrypt the sealed secret and generate a Kubernetes secret from it. Your secret got successfully created by running:

```
kubectl get secret mysecret -n ${PROJECT_NAMESPACE} -o yaml
```

6. If your sealed secret got created successfully but did not generate the matching secret, look at the logs of the controller:

```
kubectl logs -l=name=sealed-secrets-controller -n kube-system
```

7. Commit `secrets.yaml` to your repo if you have not already done so in step 3.

5.4.6.2.1.6 Remove a Secret

1. Following the same example from above in "Adding a secret", now remove the manifest for `mysecret` in `secrets.yaml` and commit those changes to the repo.
2. Delete the SealedSecret in the cluster:

```
kubectl delete SealedSecret -n ${PROJECT_NAMESPACE} mysecret
```

3. Delete the secret itself:

562 https://github.com/bitnami-labs/sealed-secrets#usage

```
kubectl delete secret -n ${PROJECT_NAMESPACE} mysecret
```

5.4.6.2.1.7 Rotate the Controller's Sealing Key

For added security, it is a good practice to rotate the key the controller uses to decrypt sealed secrets. By default, the controller generates a new key every 30 days. When this happens, you need to update the certificate you use to create sealed secrets by fetching the latest one:

```
kubeseal --fetch-cert > mycert.pem
```

Do not forget to commit it back to the repo!

In a disaster case, let's say your cluster gets destroyed, you would lose all your sealing keys, so you would not be able to recreate all the secrets from the sealed secrets in your GitOps repo. For this reason, you might want to back up the sealing keys. To do this every time a new sealing key is generated, run:

```
kubectl get secret -n kube-system -l sealedsecrets.bitnami.com/sealed-secrets-key -o
yaml > sealing-key
```

Then store `sealing-key` with the others in a safe location such as OneLogin Notes or Vault. To restore from a backup after a disaster, recreate all of the sealing keys with `kubectl apply -f sealing-key1 sealing-key2 ...` before starting the controller. If the controller was already started, restart it: `kubectl delete pod -n kube-system -l name=sealed-secrets-controller`

To disable sealing key rotation For example, configure the controller's command in the pod template with `--key-renew-period=0`. See the following YAML file.

```
Pod Template:
  Labels:           name=sealed-secrets-controller
  Service Account:  sealed-secrets-controller
  Containers:
   sealed-secrets-controller:
    Image:      docker.io/bitnami/sealed-secrets-controller:v0.18.1
    Port:       8080/TCP
    Host Port:  0/TCP
    Command:    controller
      --key-renew-period=0
```

If required, edit the controller's manifest with:

```
kubectl edit deployment.apps/sealed-secrets-controller -n kube-system
```

5.4.6.2.2 Deploy a Sample App from DKP GitOps

Enterprise Gov Advanced

Use this procedure to deploy a sample `podinfo` application from DKP Enterprise GitOps.

5.4.6.2.2.1 Prerequisites

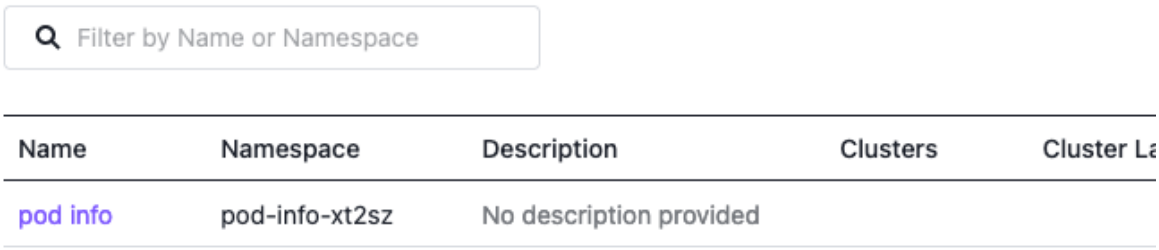
- Enterprise DKP installed (see page 127)
- Kommander installed (see page 1905)
- Github account and personal access token[563]
- Add cluster to Kommander (see page 784)
- Setup Workspace and Projects (see page 678)

5.4.6.2.2.2 Deployment Steps

> This procedure was run on an AWS cluster with DKP installed.

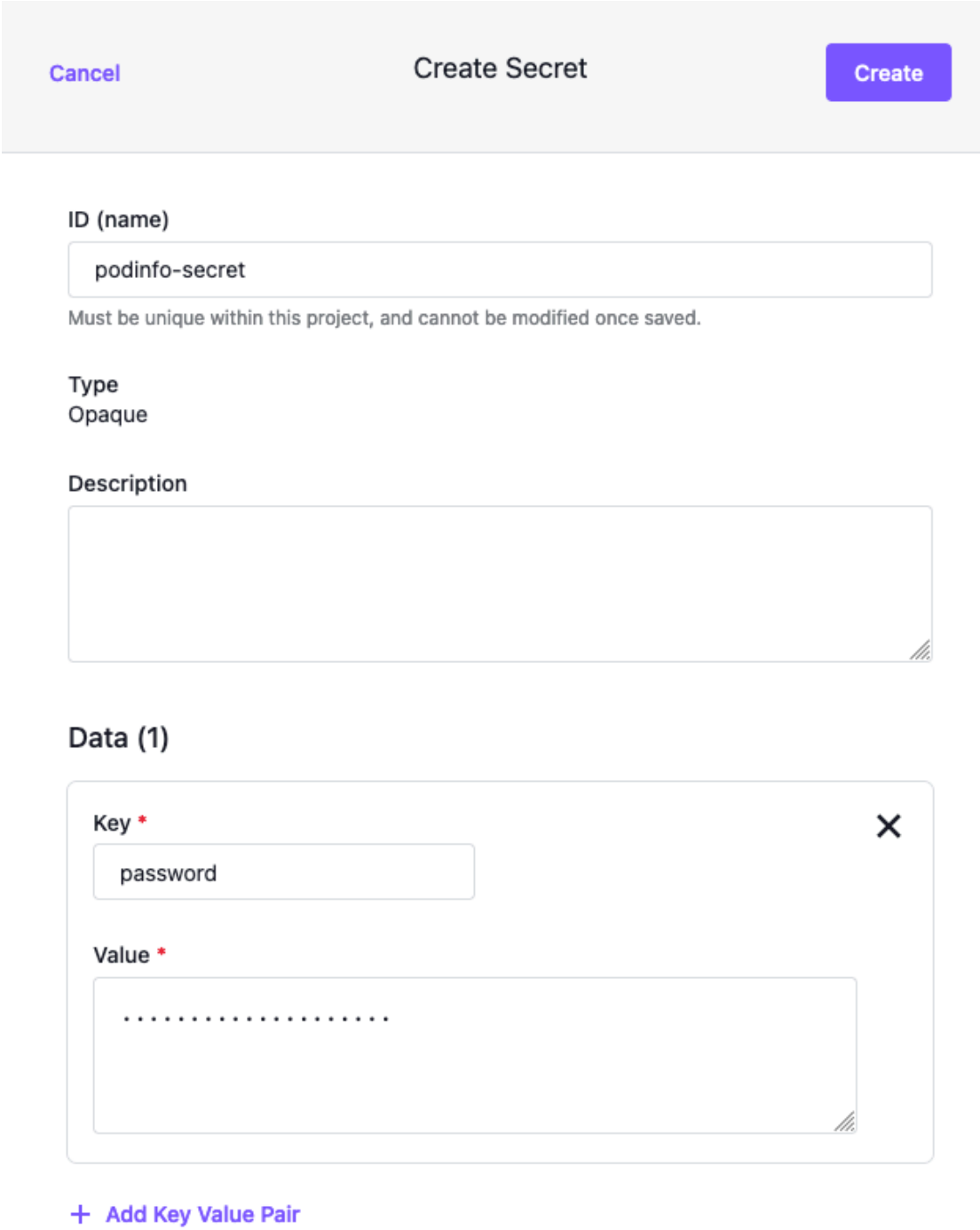
Follow these steps:

1. Ensure you are on the **Default Workspace** (or other workspace you have access to) so that you can create a project.
2. Create a project (see page 716).
   In the working example we name the project **pod-info**. When you create a namespace, Kommander appends five alphanumeric characters. You can opt to select a target cluster for this project from one of the available attached clusters, and then this **(pod-info-xxxxx)** is the namespace used for deployments under the project, for example:



563 https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

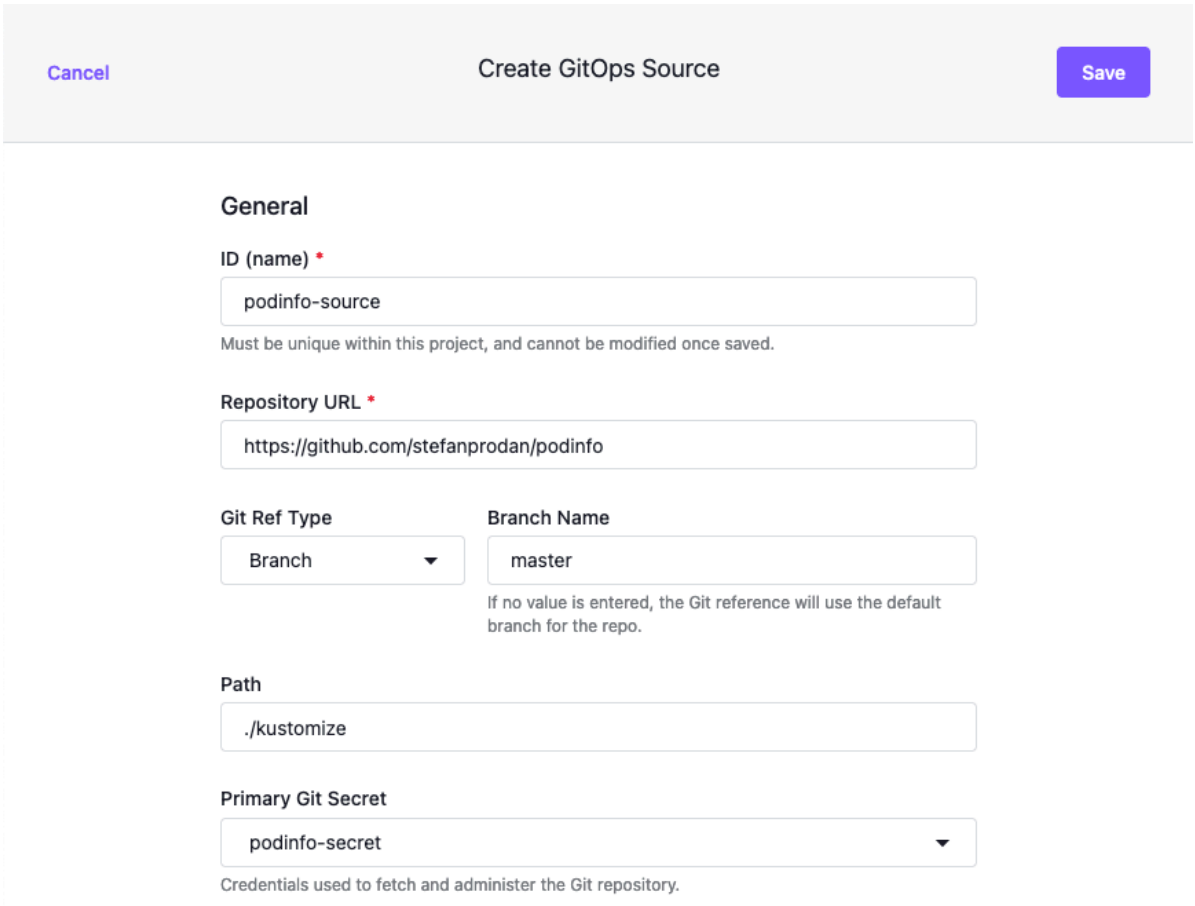
3. [Optional] Create a secret in order to pull from the repository, for private repositories.
    a. Select the Secrets tab and set up your secret according to the Continuous Deployment documentation (see page 750).
    b. Add a **key** and **value** pair for the GitHub personal access token and then select **Create**.



4. Verify that the secret `podinfo-secret` is created on the project namespace in the **managed or attached** cluster:

```
kubectl get secrets -n pod-info-xt2sz --kubeconfig=${CLUSTER_NAME}.conf
```

```
NAME                          TYPE                                  DATA   AGE
default-token-k685t           kubernetes.io/service-account-token   3      94m
pod-info-xt2sz-token-p9k5z    kubernetes.io/service-account-token   3      94m
podinfo-secret                Opaque                                1      1s
tls-root-ca                   Opaque                                1      93m
```

5. Select your project and then select the **CD** tab.
6. Add a GitOps Source, complete the required fields, and then **Save**.
   There are several configurable options such as selecting the **Git Ref Type** but in this example we use the master branch. The **Path** value should contain where the manifests are located. Additionally, the **Primary Git Secret** is the secret (**podinfo-secret**) that you created in the previous step, if you need to access private repositories. This can be disregarded for public repositories.
7. 

a. Verify the status of `gitrepository` creation with this command (on the attached or managed cluster), and if **READY** is marked as **True**:

```
kubectl get gitrepository -A --kubeconfig=${CLUSTER_NAME}.conf
```

```
NAMESPACE        NAME           URL
AGE    READY   STATUS
kommander-flux   management     https://gitea-http.kommander.svc/kommander/
kommander.git   134m   True    stored artifact for revision 'main/
4fbee486076778c85e14f3196e49b8766e50e6ce'
pod-info-xt2sz   podinfo-source https://github.com/stefanprodan/podinfo
116m   True    stored artifact for revision 'master/
b3b00fe35424a45d373bf4c7214178bc36fd7872'
```

8. Verify the `Kustomization` with this command below (on the attached or managed cluster), and if **READY** is marked as **True**:

```
kubectl get kustomizations -n pod-info-xt2sz --kubeconfig=${CLUSTER_NAME}.conf
```

```
NAME                  AGE    READY    STATUS
originalpodinfo       10m    True     Applied revision: master/
b3b00fe35424a45d373bf4c7214178bc36fd7872
podinfo-source        113m   True     Applied revision: master/
b3b00fe35424a45d373bf4c7214178bc36fd7872
project               116m   True     Applied revision: main/
4fbee486076778c85e14f3196e49b8766e50e6ce
project-tls-root-ca   117m   True     Applied revision: main/
4fbee486076778c85e14f3196e49b8766e50e6ce
```

Note the **port** so that you can use to verify if the app is deployed correctly (on the attached or managed cluster):

```
kubectl get deployments,services -n pod-info-xt2sz --kubeconfig=$
{CLUSTER_NAME}.conf
```

```
NAME                       READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/podinfo    2/2      2             2            118m
```

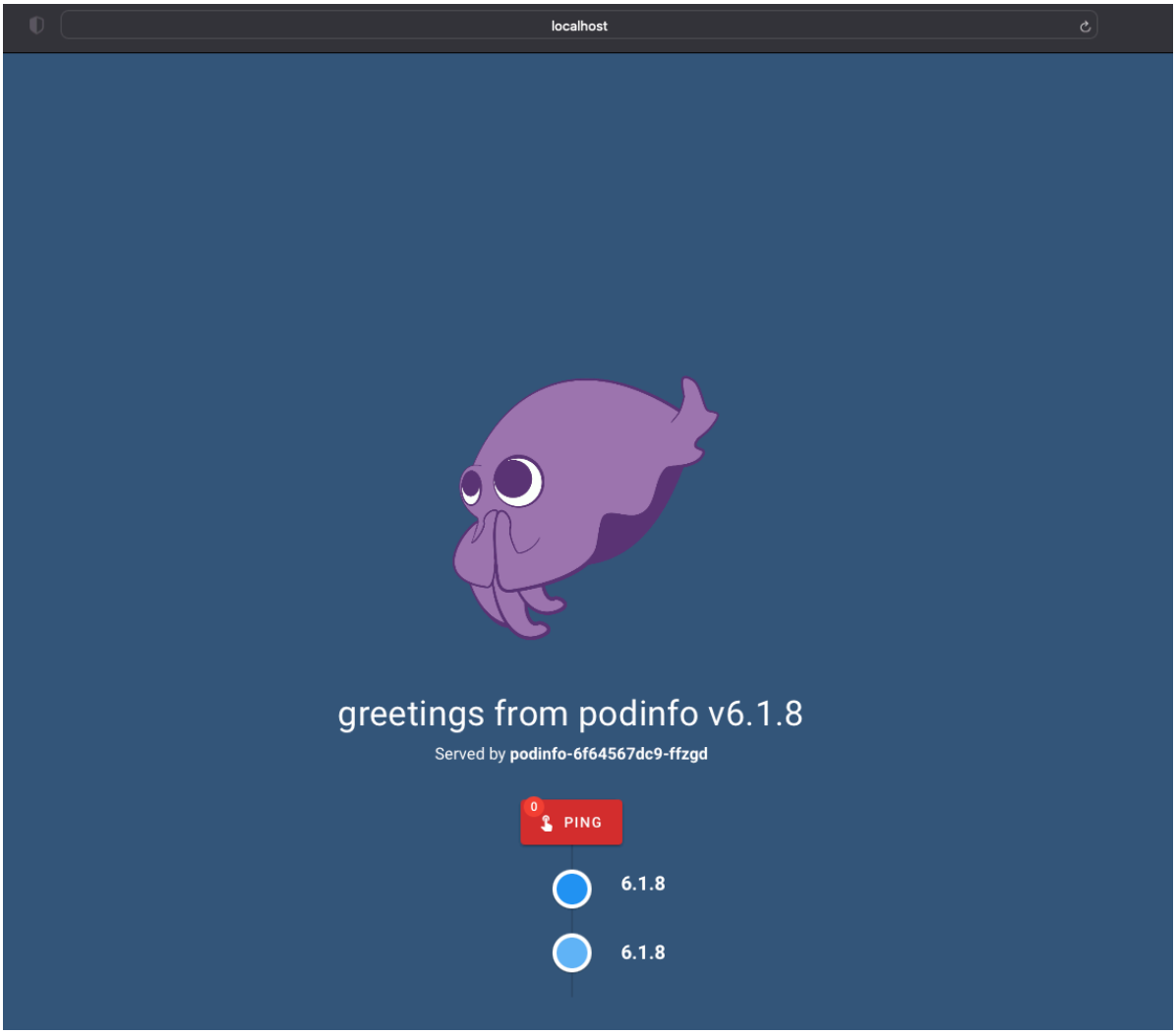
```
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/podinfo   ClusterIP   10.99.239.120   <none>        9898/TCP,9999/TCP
118m
```

9. Port forward the **podinfo** service (port **9898**) to verify (on the attached or managed cluster):

```
kubectl port-forward svc/podinfo -n pod-info-xt2sz 9898:9898 --kubeconfig=$
{CLUSTER_NAME}.conf
```

```
Forwarding from 127.0.0.1:9898 -> 9898
Forwarding from [::1]:9898 -> 9898
Handling connection for 9898
Handling connection for 9898
Handling connection for 9898
```

10. Open a browser and type in **localhost:9898**. A successful deployment of the podinfo app gives you this page:



5.4.6.3 Continuous Deployment

Enterprise Gov Advanced

After installing Kommander and configuring your project and its clusters (see page 774), navigate to the **Continuous Deployment (CD)** tab under your Project. Here you create a GitOps source which is a source code management (SCM) repository hosting the application definition. D2iQ recommends that you create a secret first then create a GitOps source accessed by the secret.

5.4.6.3.1 Prerequisites

You must have Kommander installed (see page 1558) to run this procedure.

5.4.6.3.2 Set up a Secret for Accessing GitOps

Create a secret that Kommander uses to deploy the contents of your GitOps repository:

> This dialog box creates a `types.kubefed.io/v1beta1, Kind=FederatedSecret` and this is not yet supported by DKP CLI. Use the GUI, as described above, to create a federated secret or create a `FederatedSecret` manifest and apply it to the project namespace. Learn more about FederatedSecrets (see page 765).

Kommander secrets (for CD) can be configured to support any of the following three authentication methods:

- HTTPS Authentication (described above)
- HTTPS self-signed certificates
- SSH Authentication

The following table describes the fields required for each authentication method:

HTTP Auth	HTTPS Auth (Self-signed)	SSH Auth
username	username	identity
password	password	identity.pub
	caFile	known_hosts

If you are using GitOps by using a GitHub repo as your source, you can create your secret with a personal access token[564]. Then, in the DKP UI, in your project, create a Secret, with a key:value pair of `password` : `<your-token-created-on-github>` . If you are using a GitHub personal access token, you do not need to have a key:value pair of `username` : `<your-github-username>` .

If you are using a secret with your GitHub username and your password, you will need one secret created in the DKP UI, with key:value pairs of `username` : `<your-github-username>` and `password` : `<your-github-password>` . **Note:** if you have multi-factor authentication turned on in your GitHub account, this will not work.

564 https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

Using a token without a username is valid for GitHub, but other providers (such as GitLab) require both username and tokens.

If you are using a public GitHub repository, you do not need to use a secret.

5.4.6.3.3 Create GitOps Source

After the secret is created, you can view it in the `Secrets` tab. Configure the GitOps source accessed by the secret.

If using an SSH secret, the SCM repo URL needs to be an SSH address. It does not support SCP syntax. The URL format is `ssh://user@host:port/org/repository`.

It takes a few moments for the GitOps Source to be reconciled and the manifests from the SCM repository at the given path to be federated to attached clusters. After the sync is complete, manifests from GitOps source are created in attached clusters.

After a GitOps Source is created, there are various commands that can be executed from the CLI to check the various stages of syncing the manifests.

On the management cluster, check your `GitopsRepository` to ensure that the `CD` manifests have been created successfully.

```
kubectl describe gitopsrepositories.dispatch.d2iq.io -n<PROJECT_NAMESPACE> gitopsdemo
```

```
Name:         gitopsdemo
Namespace:    <PROJECT_NAMESPACE>
...
Events:
  Type    Reason               Age    From                         Message
  ----    ------               ----   ----                         -------
  Normal  ManifestSyncSuccess  1m7s  GitopsRepositoryController  manifests synced to
bootstrap repo
...
```

On the attached cluster, check for your `Kustomization` and `GitRepository` resources. The `status` field reflects the syncing of manifests:

```
kubectl get kustomizations.kustomize.toolkit.fluxcd.io -n<PROJECT_NAMESPACE>
<GITOPS_SOURCE_NAME> -oyaml
```

```
...
status:
  conditions:
  - reason: ReconciliationSucceeded
    status: "True"
    type: Ready
    ...
...
```

Similarly, with `GitRepository` resource:

```
kubectl get gitrepository.source.toolkit.fluxcd.io -n<PROJECT_NAMESPACE>
<GITOPS_SOURCE_NAME> -oyaml
```

```
...
status:
  conditions:
  - reason: GitOperationSucceed
    status: "True"
    type: Ready
    ...
...
```

If there are errors creating the manifests, those events are populated in the status field of the `GitopsRepository` resource on the management cluster, the `GitRepository` and `Kustomization` resources on the attached cluster(s), or both.

5.4.6.3.4 Suspend GitOps Source

There may be times when you need to suspend the auto-sync between the GitOps repository and the associated clusters. This *live debugging* may be necessary to resolve an incident in the minimum amount of time without the overhead of pull request based workflows.

To Suspend the GitOps Source from the DKP UI:

1. From the top menu bar, select your target workspace.
2. Select **Projects** from the sidebar menu.
3. Select your project from the list.
4. Select the **Continuous Deployment (CD)** tab.
5. Select the three dot button to the right of the desired GitOps Source.

6. Select **Suspend** to manually suspend the GitOps reconciliation.

This lets you use `kubectl`, `helm`, or another tool to resolve the issue. After the issue is resolved select **Resume** to sync the updated contents of the GitOps source to the associated clusters.

Similar to **Suspend/Resume**, you can use the **Delete** action to remove the GitOps source. Removing the GitOps source results in removal of all the manifests applied from the GitOps source.

You can have more than one GitOps Source in your Project to deploy manifests from various sources.

Kommander deployments are backed by FluxCD. Please refer to Flux Source Controller[565] and Kustomize controller[566] docs for advanced configuration and more examples.

5.4.6.4 Project Deployments Troubleshooting

Enterprise Gov Advanced

- Events related to federation are stored in respective `FederatedGitRepository`, `FederatedKustomization`, or both resources.
- View the events and logs for `deployments/Kommander-repository-controller` in Kommander namespace, if there are any unexpected errors.
- Enabling the Kommander repository controller for your project namespace causes a number of related Flux controller components[567] to deploy into the namespace. These are necessary for the proper operation of the repository controller and should not be removed.
- Ensure your GitOps repository does not contain any manifests that are cluster-scoped - for example, `Namespace`, `ClusterRole`, `ClusterRoleBinding`, etc. All of the manifests must be namespace-scoped.
- Ensure your GitOps repository does not contain any `HelmRelease` and `Kustomization` resources that are targeting a different namespace than the project namespace.

5.4.6.5 View Helm Releases

Enterprise Gov Advanced

5.4.6.5.1 View Helm Releases for Continuous Deployments

In addition to viewing the current GitOps Sources, you can also view the currently deployed Helm Releases.

1. From the top menu bar, select your target workspace.
2. Select **Projects** from the sidebar menu.
3. Select your project from the list.

565 https://fluxcd.io/docs/components/source/
566 https://fluxcd.io/docs/components/kustomize/
567 https://toolkit.fluxcd.io/components/

4. Select the **Continuous Deployment (CD)** tab.
5. Select the **Helm Releases** button.

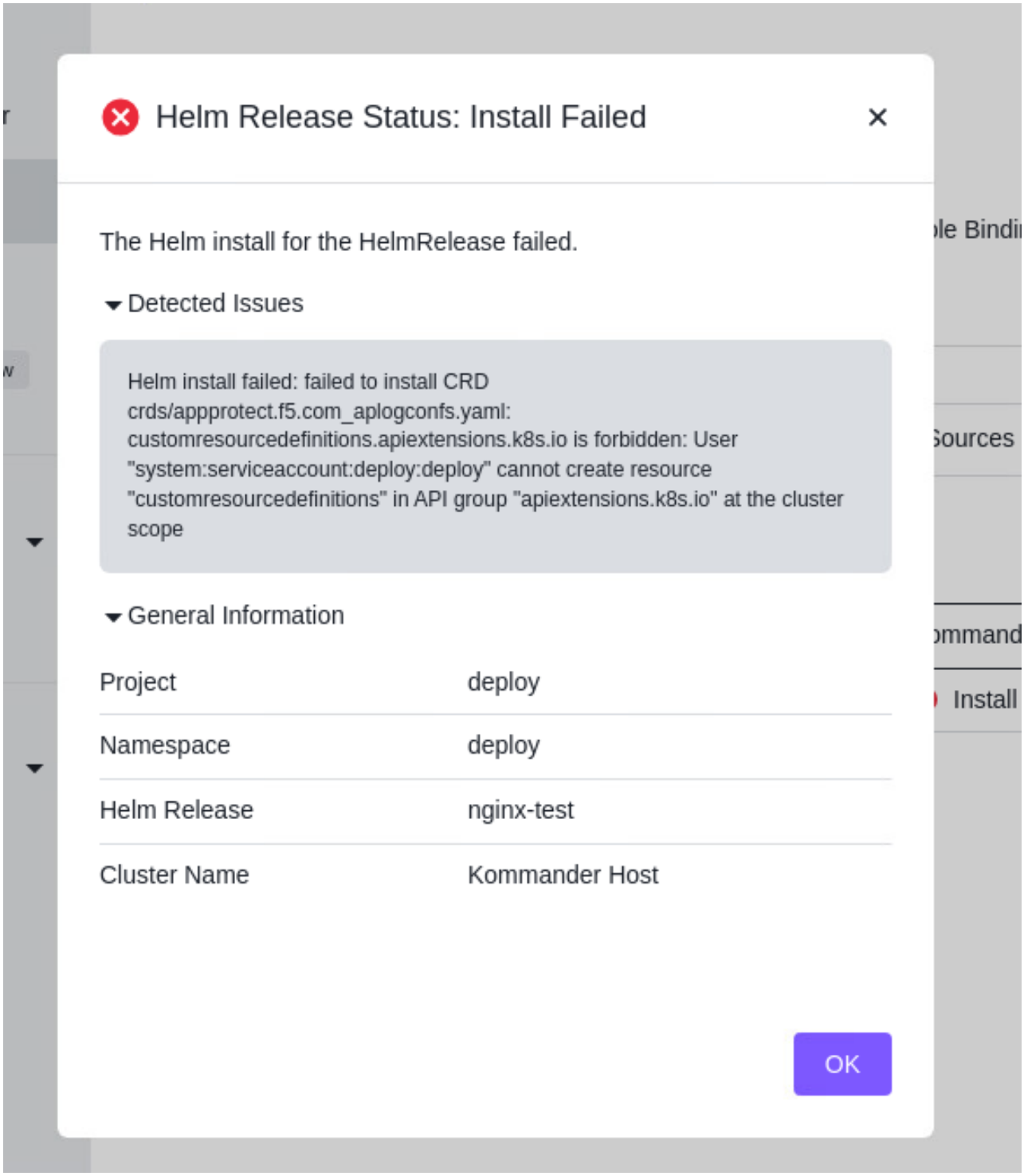
All of the current Helm Release charts are displayed with their Chart Version and the names of the clusters. In this example *daily* is the name of the current cluster being displayed.

5.4.6.5.1.1 Troubleshooting

If an error occurs with the Helm Releases charts deployment, an “Install Failed” error status appears in the **Kommander Host** Field

- Select the error status to open a screen that details specific issues related to the error.

Example:



5.4.6.5.1.2 Edit GitOps Source

To edit the GitOPs Source from within the UI, follow these steps:

1. From the top menu bar, select your target workspace.
2. Select **Projects** from the sidebar menu.
3. Select your project from the list.
4. Select the **Continuous Deployment (CD)** tab.
5. Select the **GitOps Sources** button.

From here, you can edit the **ID (name)**, **Repository URL**, **Git Ref Type** , **Branch Name**, **Type**, **Path**, and **Primary Git Secret**.

5.4.7 Project Role Bindings

Enterprise Gov Advanced

Project Role Bindings grant access to a specified Project Role for a specified group of people

5.4.7.1 Configure Project Role Bindings - UI Method

Before you can create a Project Role Binding, ensure you have created a Group. A Kommander Group can contain one or several Identity Provider users or groups.

You can assign a role to this Kommander Group:

1. From the **Projects** page, select your project.
2. Select the **Role Bindings** tab, then select **Add Roles** next to the group you want.
3. Select the **Role**, or Roles, you want from the drop-down menu, and then select **Save**.

5.4.7.2 Configure Project Role Bindings - CLI Method

A Project role binding can also be created using kubectl:

```
cat << EOF | kubectl create -f -
apiVersion: workspaces.kommander.mesosphere.io/v1alpha1
kind: VirtualGroupProjectRoleBinding
metadata:
  generateName: projectpolicy-
  namespace: ${projectns}
spec:
  projectRoleRef:
    name: ${projectrole}
  virtualGroupRef:
```

```
    name: ${virtualgroup}
EOF
```

5.4.7.3 Configure Project Role Bindings to Bind to WorkspaceRoles - CLI Method

You can also create a Project role binding to bind to a WorkspaceRole in certain instances. To list the WorkspaceRoles that you can bind to a Project, run the following command:

```
kubectl get workspaceroles -n ${workspacens} -o=jsonpath="{.items[?
(@.metadata.annotations.workspace\.kommander\.d2iq\.io\/project-default-workspace-
role-for==\"${projectns}\")].metadata.name}"
```

You can bind to any of the above WorkspaceRoles by setting `spec.workspaceRoleRef` in the project role binding:

```
cat << EOF | kubectl create -f -
apiVersion: workspaces.kommander.mesosphere.io/v1alpha1
kind: VirtualGroupProjectRoleBinding
metadata:
  generateName: projectpolicy-
  namespace: ${projectns}
spec:
  workspaceRoleRef:
    name: ${workspacerole}
  virtualGroupRef:
    name: ${virtualgroup}
EOF
```

Note that you must specify either workspaceRoleRef or projectRoleRef to be validated by the admission webhook. Specifying both values is not valid and will cause an error.

Ensure the `projectns`, `workspacens`, `projectrole` (or `workspacerole`) and the `virtualgroup` variables are set before executing the command.

When a Project Role Binding is created, Kommander creates a Kubernetes `FederatedRoleBinding` on the Kubernetes cluster where Kommander is running. You can view this by first finding the name of the project role binding that you created: `kubectl -n ${projectns} get federatedrolebindings.types.kubefed.io`.

Then, view the details like in this example:

```
kubectl -n ${projectns} get federatedrolebindings.types.kubefed.io projectpolicy-
gtct4-rdkwq -o yaml
```

Output:

```
apiVersion: types.kubefed.io/v1beta1
kind: FederatedRoleBinding
metadata:
  creationTimestamp: "2020-06-04T16:19:27Z"
  finalizers:
  - kubefed.io/sync-controller
  generation: 1
  name: projectpolicy-gtct4-rdkwq
  namespace: project1-5ljs9-lhvjl
  ownerReferences:
  - apiVersion: workspaces.kommander.mesosphere.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: VirtualGroupProjectRoleBinding
    name: projectpolicy-gtct4
    uid: 19614de2-4593-433e-82fa-96dc9470e07a
  resourceVersion: "196270"
  selfLink: /apis/types.kubefed.io/v1beta1/namespaces/project1-5ljs9-lhvjl/
federatedrolebindings/projectpolicy-gtct4-rdkwq
  uid: beaffc29-edec-4258-9813-3a17ba27a2a6
spec:
  placement:
    clusterSelector: {}
  template:
    roleRef:
      apiGroup: rbac.authorization.k8s.io
      kind: Role
      name: admin-dbfpj-l6s9g
    subjects:
    - apiGroup: rbac.authorization.k8s.io
      kind: User
      name: user1@d2iq.lab
status:
  clusters:
  - name: konvoy-5nr5h
  conditions:
  - lastTransitionTime: "2020-06-04T16:19:27Z"
    lastUpdateTime: "2020-06-04T16:19:27Z"
    status: "True"
    type: Propagation
  observedGeneration: 1
```

Then, if you run the following command on a Kubernetes cluster associated with the Project, you'll see a Kubernetes RoleBinding Object, in the corresponding namespace:

```
kubectl -n ${projectns} get rolebinding projectpolicy-gtct4-rdkwq -o yaml
```

Output:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  creationTimestamp: "2020-06-04T16:19:27Z"
  labels:
    kubefed.io/managed: "true"
  name: projectpolicy-gtct4-rdkwq
  namespace: project1-5ljs9-lhvjl
  resourceVersion: "125392"
  selfLink: /apis/rbac.authorization.k8s.io/v1/namespaces/project1-5ljs9-lhvjl/
rolebindings/projectpolicy-gtct4-rdkwq
  uid: 2938398d-437b-4f3a-9cb9-c92e50139196
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: admin-dbfpj-l6s9g
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: user1@d2iq.lab
```

5.4.7.4 Role Binding with VirtualGroup

In DKP, `VirtualGroup` is a list of subjects that can be assigned to several different kinds of roles, including:

- `ClusterRole` for cluster-scoped objects
- `WorkspaceRole` for workspace-scoped objects
- `ProjectRole` for project-scoped objects

In order to define which `VirtualGroup` (s) is assigned to one of these roles, administrators can create corresponding role bindings such as `VirtualGroupClusterRoleBinding`, `VirtualGroupWorkspaceRoleBinding`, and `VirtualGroupProjectRoleBinding`.

For more information about configuring `VirtualGroup`, please refer to the DKP API Documentation (see page 1815).

Note that for `WorkspaceRole` and `ProjectRole`, the referenced `VirtualGroup` and corresponding role and role binding objects need to be in the same namespace. If they are not in the same namespace, the role will not bind to the `VirtualGroup` since it is assumed that the rules set in the role apply to objects that live in that namespace. Whereas for `ClusterRole` which is cluster-scoped, the `VirtualGroupClusterRoleBinding` is also cluster-scoped, even though it references a namespace-scoped `VirtualGroup`.

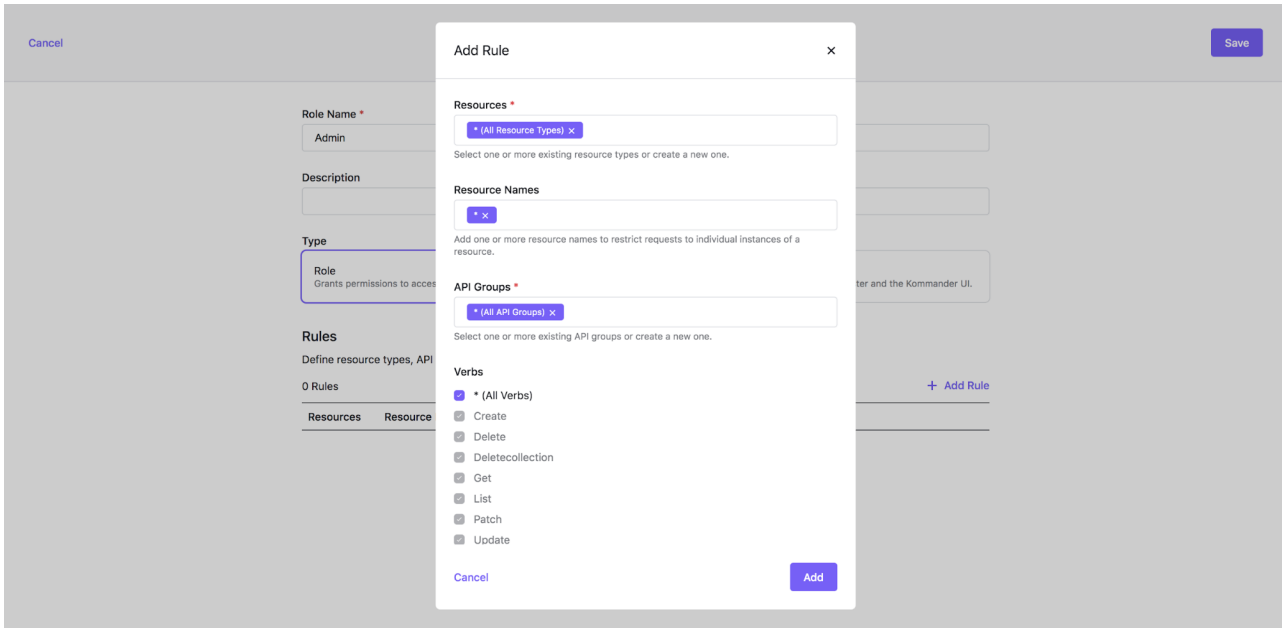
5.4.8 Project Roles

Enterprise Gov Advanced

Project Roles are used to define permissions at the namespace level.

5.4.8.1 Configure Project Role - UI Method

In the example below, a Project Role is created with a single Rule. This Project Role corresponds to an admin role.



You can create a Project Role with several Rules.

5.4.8.2 Configure Project Role - CLI Method

The same Project Role can also be created using kubectl:

```
cat << EOF | kubectl create -f -
apiVersion: workspaces.kommander.mesosphere.io/v1alpha1
kind: ProjectRole
metadata:
  annotations:
    kommander.mesosphere.io/display-name: Admin
  generateName: admin-
  namespace: ${projectns}
spec:
  rules:
  - apiGroups:
```

```
    - '*'
    resources:
    - '*'
    verbs:
    - '*'
EOF
```

Ensure the `projectns` variable is set before executing the command.

You can set it using the following command (for a Kommander Project called `project1`, and after setting the `workspacens` as explained in the previous section):

```
projectns=$(kubectl -n ${workspacens} get projects.workspaces.kommander.mesosphere.io
-o jsonpath='{.items[?
(@.metadata.generateName=="project1-")].status.namespaceRef.name}')
```

When a Project Role is created, Kommander creates a Kubernetes `FederatedRole` on the Kubernetes cluster where Kommander is running:

```
kubectl -n ${projectns} get federatedroles.types.kubefed.io admin-dbfpj-l6s9g -o yaml
apiVersion: types.kubefed.io/v1beta1
kind: FederatedRole
metadata:
  creationTimestamp: "2020-06-04T11:54:26Z"
  finalizers:
  - kubefed.io/sync-controller
  generation: 1
  name: admin-dbfpj-l6s9g
  namespace: project1-5ljs9-lhvjl
  ownerReferences:
  - apiVersion: workspaces.kommander.mesosphere.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ProjectRole
    name: admin-dbfpj
    uid: e5f3b2ca-16bf-474d-8305-7be04c034793
  resourceVersion: "75680"
  selfLink: /apis/types.kubefed.io/v1beta1/namespaces/project1-5ljs9-lhvjl/
federatedroles/admin-dbfpj-l6s9g
  uid: 1e5a3d98-b223-4605-bba1-16276a3eb47c
spec:
  placement:
    clusterSelector: {}
  template:
    rules:
    - apiGroups:
      - '*'
      resourceNames:
      - '*'
```

```
      resources:
      - '*'
      verbs:
      - '*'
status:
  clusters:
  - name: konvoy-5nr5h
  conditions:
  - lastTransitionTime: "2020-06-04T11:54:26Z"
    lastUpdateTime: "2020-06-04T11:54:26Z"
    status: "True"
    type: Propagation
  observedGeneration: 1
```

Then, if you run the following command on a Kubernetes cluster associated with the Project, you see a Kubernetes Role object in the corresponding namespace:

```
kubectl -n ${projectns} get role admin-dbfpj-l6s9g -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  creationTimestamp: "2020-06-04T11:54:26Z"
  labels:
    kubefed.io/managed: "true"
  name: admin-dbfpj-l6s9g
  namespace: project1-5ljs9-lhvjl
  resourceVersion: "29218"
  selfLink: /apis/rbac.authorization.k8s.io/v1/namespaces/project1-5ljs9-lhvjl/roles/
admin-dbfpj-l6s9g
  uid: f05b998c-4649-4e73-bbfe-c12bc4c86a3c
rules:
- apiGroups:
  - '*'
  resourceNames:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
```

5.4.9 Project ConfigMaps

Enterprise Gov Advanced

Use ConfigMaps to automate ConfigMaps aretion on your clusters

Project ConfigMaps can be created to make sure Kubernetes ConfigMaps are automatically created on all Kubernetes clusters associated with the Project, in the corresponding namespace.

As reference, a ConfigMap is a key-value pair to store some type of non-confidential data like “name=bob” or “state=CA”. For a full reference to the concept, consult the Kubernetes documentation on the topic of ConfigMaps[568].

5.4.9.1 Configuring Project ConfigMaps - UI Method

The below Project ConfigMap form can be navigated to by:

1. From the top menu bar, select your target workspace.
2. Select **Projects** from the sidebar menu.
3. Select your project from the list.
4. Select the **ConfigMaps** tab to browse the deployed ConfigMaps.
5. Select **+ Create ConfigMap** button.
6. Enter an ID, Description and Data for the ConfigMap, and select the **Create** button.

5.4.9.2 Configuring Project ConfigMaps - CLI Method

A Project ConfigMap is simply a Kubernetes FederatedConfigMap and can be created using kubectl with YAML:

```
cat << EOF | kubectl create -f -
apiVersion: types.kubefed.io/v1beta1
kind: FederatedConfigMap
metadata:
  generateName: cm1-
  namespace: ${projectns}
spec:
  placement:
    clusterSelector: {}
  template:
    data:
      key: value
EOF
```

Ensure the `projectns` variable is set before executing the command. This variable is the project namespace (the Kubernetes Namespace associated with the project) that was defined/created when the project itself was initially created.

```
projectns=$(kubectl -n ${workspacens} get projects.workspaces.kommander.mesosphere.io
-o jsonpath='{.items[?
(@.metadata.generateName=="project1-")].status.namespaceRef.name}')
```

568 https://kubernetes.io/docs/concepts/configuration/configmap/

Then, if you run the following command on a Kubernetes cluster associated with the Project, you'll see a Kubernetes ConfigMap Object, in the corresponding namespace:

```
kubectl -n ${projectns} get configmap cm1-8469c -o yaml
apiVersion: v1
data:
  key: value
kind: ConfigMap
metadata:
  creationTimestamp: "2020-06-04T16:37:10Z"
  labels:
    kubefed.io/managed: "true"
  name: cm1-8469c
  namespace: project1-5ljs9-lhvjl
  resourceVersion: "131844"
  selfLink: /api/v1/namespaces/project1-5ljs9-lhvjl/configmaps/cm1-8469c
  uid: d32acb98-3d57-421f-a677-016da5dab980
```

5.4.10 Project Secrets

Enterprise Gov Advanced

Project Secrets can be created to make sure a Kubernetes Secrets are automatically created on all the Kubernetes clusters associated with the Project, in the corresponding namespace.

5.4.10.1 Configure Project Secrets - UI Method

1. Select the workspace your project was created in from the workspace selection dropdown in the header.
2. In the sidebar menu, select **Projects**.
3. Select the project you want to configure from the table.
4. Select the **Secrets** tab, and then select the **Create Secret** button.
5. Complete the form and select the **Create** button.

5.4.10.2 Configure Project Secrets - CLI Method

A Project Secret is simply a Kubernetes FederatedConfigSecret and can also be created using kubectl:

```
cat << EOF | kubectl create -f -
apiVersion: types.kubefed.io/v1beta1
kind: FederatedSecret
metadata:
  generateName: secret1-
```

```
  namespace: ${projectns}
spec:
  placement:
    clusterSelector: {}
  template:
    data:
      key: dmFsdWU=
EOF
```

Ensure the `projectns` variable is set before executing the command.

```
projectns=$(kubectl -n ${workspacens} get projects.workspaces.kommander.mesosphere.io
-o jsonpath='{.items[?
(@.metadata.generateName=="project1-")].status.namespaceRef.name}')
```

The value of the key is base64 encoded.

If you run the following command on a Kubernetes cluster associated with the Project, you see a Kubernetes Secret Object, in the corresponding namespace:

```
kubectl -n ${projectns} get secret secret1-r9vk2 -o yaml
apiVersion: v1
data:
  key: dmFsdWU=
kind: Secret
metadata:
  creationTimestamp: "2020-06-04T16:51:59Z"
  labels:
    kubefed.io/managed: "true"
  name: secret1-r9vk2
  namespace: project1-5ljs9-lhvjl
  resourceVersion: "137215"
  selfLink: /api/v1/namespaces/project1-5ljs9-lhvjl/secrets/secret1-r9vk2
  uid: e5c6fc1d-93e7-47fe-ae1e-f418f8e35d72
type: Opaque
```

5.4.11 Project Quotas & Limit Ranges

Enterprise Gov Advanced

Project Quotas and Limit Ranges can be set up to limit the number of resources the Project team uses.

5.4.11.1 Creating Project Quotas & Limit Ranges- UI Method

Project Quotas and Limit Ranges can be set up to limit the number of resources the Project team uses. Quotas and Limit Ranges are applied to all project clusters.

1. Select the workspace your project was created in from the workspace selection dropdown in the header.
2. In the sidebar menu, select **Projects**.
3. Select the project you want to configure from the table.
4. Select the **Quotas & Limit Ranges** tab, and then select the **Edit** button.

   Kommander provides a set of default resources for which you can set Quotas. You can also define Quotas for custom resources. We recommend that you set Quotas for CPU and Memory. By using Limit Ranges, you can restrict the resource consumption of individual Pods, Containers, and Persistent Volume Claims in the project namespace. You can also constrain memory and CPU resources consumed by Pods and Containers, and storage resources consumed by Persistent Volume Claims.
5. To add a custom quota, scroll to the bottom of the form and select **Add Quota**.
6. When you are finished, select the **Save** button.

5.4.11.2 Create Project Quotas & Limit Ranges - CLI Method

All the Project Quotas are defined using a Kubernetes FederatedResourceQuota called `kommander` which you can also create/update using kubectl:

```
cat << EOF | kubectl apply -f -
apiVersion: types.kubefed.io/v1beta1
kind: FederatedResourceQuota
metadata:
  name: kommander
  namespace: ${projectns}
spec:
  placement:
    clusterSelector: {}
  template:
    spec:
      hard:
        limits.cpu: "10"
        limits.memory: 1024.000Mi
EOF
```

Ensure the `projectns` variable is set before executing the command.

```
projectns=$(kubectl -n ${workspacens} get projects.workspaces.kommander.mesosphere.io
-o jsonpath='{.items[?
(@.metadata.generateName=="project1-")].status.namespaceRef.name}')
```

Then, if you run the following command on a Kubernetes cluster associated with the Project, you'll see a Kubernetes Secret Object in the corresponding namespace:

```
kubectl -n ${projectns} get resourcequota kommander -o yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  creationTimestamp: "2020-06-05T08:04:37Z"
  labels:
    kubefed.io/managed: "true"
  name: kommander
  namespace: project1-5ljs9-lhvjl
  resourceVersion: "470822"
  selfLink: /api/v1/namespaces/project1-5ljs9-lhvjl/resourcequotas/kommander
  uid: 925b61b4-134b-4c45-915c-96a05b63d3c3
spec:
  hard:
    limits.cpu: "10"
    limits.memory: 1Gi
status:
  hard:
    limits.cpu: "10"
    limits.memory: 1Gi
  used:
    limits.cpu: "0"
    limits.memory: "0"
```

Similarly, Project Limit Ranges are defined using a FederatedLimitRange object with name `kommander` in the project namespace:

```
cat << EOF | kubectl apply -f -
apiVersion: types.kubefed.io/v1beta1
kind: FederatedLimitRange
metadata:
  name: kommander
  namespace: ${projectns}
spec:
  placement:
    clusterSelector: {}
  template:
    spec:
      limits:
      - type: "Pod"
        max:
          cpu: 500m
```

```
          memory: 50Gi
        min:
          cpu: 100m
          memory: 10Gi
      - type: "Container"
        max:
          cpu: 2
          memory: 100Mi
        min:
          cpu: 1
          memory: 10Mi
      - type: "PersistentVolumeClaim"
        max:
          storage: 3Gi
        min:
          storage: 1Gi
EOF
```

5.4.12 Project Network Policies

Enterprise Gov Advanced

Projects are created with a secure-by-default network policy and users needing more flexibility can edit or add more policies to tailor to their unique security needs.

Cluster networking is a critical and central part of Kubernetes that can also be quite challenging. All network communication within and between clusters depends on the presence of a Container Network Interface (CNI) plugin.

5.4.12.1 About Network Plugins

Network plugins ensure that Kubernetes networking requirements are met and surface features needed by network administrators, such as enforcing network policies. Common Network Plugins include Flannel, Calico, and Weave among many others. As an example, D2iQ® Konvoy® uses the Calico CNI plugin by default, but can support others.

Since pods are short-lived, the cluster needs a way to configure the network dynamically as pods are created and destroyed. Plugins provision and manage IP addresses to interfaces and let administrators manage IPs and their assignments to containers, in addition to connections to more than one host, when needed.

5.4.12.2 About Network Policies

By default, and to enable fluid communications within and between clusters, all traffic is authorized between nodes and pods. Most production environments require some kind of traffic flow control at the IP address or port level. An application-centric approach to this uses Network Policies. Pods are isolated by having a Network Policy that selects them, and the configuration of the policy limits the kind of traffic they can receive or send. Network policies do not conflict because they are additive. Pods selected by more than one policy are subject to the union of the policies' ingress and egress rules.

A Network Policy's rules define ingress and egress for network communications between pods and across namespaces. Successful traffic control using network policies is bi-directional. You have to configure both the egress policy on the source pod and the ingress policy on the destination pod to enable the traffic. If either end denies the traffic, it will not flow between the pods.

Since the Kubernetes default is to allow all traffic, it is a common practice to create a default "deny all traffic" rule, and then specifically open up some combination of the pods, ports, and applications as needed.

5.4.12.3 Creating Network Policies

You create network policies in three main parts:

- General information
- Ingress rules
- Egress rules

5.4.12.3.1 General information section

The fields in this part of the form allow you to create a name and description for this policy. Creating a detailed **Description** helps to keep policy functions understandable for additional use and maintenance.

This section also contains the **Pod Selector** fields for selecting pods using either Labels or Expressions. **Labels** added to pod declarations are a common means of identifying individual pods, or creating groups of pods, in a Kubernetes cluster. **Expressions** are similar to Labels, but allow you to define parameters that identify a range of pods.

The **Policy Types** selections help to define the type of Network Policy you are creating:

- **Default** - automatically includes ingress, and egress is set only if the network policy defines egress rules.
- **Ingress** - this policy applies to ingress traffic for the selected pods, to namespaces using the options you define below, or both.
- **Egress** - this policy applies to egress traffic for the selected pods, to namespaces using the options you define below, or both.

If the Default policy type is too rigid or does not offer what you need, you can select the Ingress or Egress type, or both, and explicitly define the policy with the options that follow. For example, if you do not want this policy to apply to ingress traffic, you would select only Egress, and then define the policy.

To deny all ingress traffic, select the **Ingress** option here and then leave the ingress rules empty.

To deny all egress traffic, select the **Egress** option here and then leave the egress rules empty.

5.4.12.3.2 Ingress rules section

Ingress rules use a combination of **Port** / **Protocol** and **Source** to define the incoming traffic allowed to some or all of the pods in this namespace.

The options under **Sources: From** enable you to define a source either by using the pod selector or by defining an IP block. When using the pod selector method, you can define the namespace, the pods within that namespace, or both.

Namespaces - Selecting a namespace in an ingress rule source permits the pods selected by the pod selector, in your selected namespaces, to receive incoming traffic that meets the other defined criteria. If you have not selected any pods, the rule permits traffic from all pods in the selected namespaces.

Pods - This option selects specific Pods which should be allowed as ingress sources or egress destinations. If you have not selected any namespaces in the namespace selector, this option selects all matching pods in the project namespace. Otherwise, this option selects all matching pods in the selected namespaces.

There also are options to select all namespaces, all pods, or both.

When defining ingress rules using the IP Block method, you define a CIDR and exception conditions. CIDR stands for Classless Inter-Domain Routing and is an IP standard for creating unique network and device identifiers. When grouped together so that they share an initial sequence of bits in their binary representation, the range of addresses creates a CIDR block. The block identity is in an IPv4-like notation including a dotted-decimal address, followed by a slash, then a colon and a number from 0 through 32, for example, 127.0.26.33:31.

5.4.12.3.3 Egress rules section

Egress rules use a similar combination of options to define the outgoing traffic from pods, ranges of pods, or namespaces in a Kommander Project. Port, Protocol, and Destination options for egress rules define the outgoing traffic. You can define your egress rules under **Destination: To**. Ensure the egress policy on the source pods, and the ingress policy on the destination pods, permit traffic in order for the pods to be able to communicate over the network.

5.4.12.4 Network Policy Examples

IMPORTANT: Before you begin each example, ensure you're on the Network Policy page for your project

To navigate to your project's Network Policy page:

1. From the top menu bar, select your target workspace.
2. Select **Projects** from the sidebar menu.
3. Select your project from the list.
4. Select the **Network Policy** tab.

5.4.12.4.1 Ingress: Permit access to API service pods from all namespaces

Suppose you need to create a network policy to permit incoming traffic to API service pods in a specific Kommander Project's namespace from any other pod in any namespace that has the label, `service.corp/users-api-role: client`. For this example, API service pods are those pods created with the Label, `service.corp/users-api-role: api`.

You can limit the policy to just incoming traffic from select namespaces by adding an ingress rule with these characteristics:

- Use Port 8080 to receive incoming TCP traffic

- Refuse traffic from pods unless they are client pods that have a specific Label, such as `service.corp/users-api-role: client`. This example follows a common microservice architecture pattern, microservice-tier-role: access_mode

5.4.12.4.1.1 Configure the general information to access API service pods

1. Select the **+ Create Network Policy** button.
2. Type “microsvc-users-api-allow” in the **ID Name** field.
3. Type “Allow Users microservice clients to reach the APIs provided in this namespace” in the **Description** field.
4. Select **Add** under **Pod Selector** and then select **Match Label**.
5. Set the **Key** to “service.corp/users-api-role” and the **Value** to “API”.

5.4.12.4.1.2 Create an Ingress rule to access API service pods

1. Leave **Policy Types** set to Default.
2. Scroll down to **Ingress Rules** and select **+ Add an Ingress Rule**.
3. Select **+ Add Port**, and set the **Port** to 8080 and the **Protocol** to TCP.

5.4.12.4.1.3 Add Sources to access API service pods

1. Select **+ Add Source** and mark the **Select All Namespaces** check box.
2. Select **+ Add Pod Selector**.
3. Select **Match Label**.
4. Set the **Key** value to “service.corp/users-api-role” and set the **Value** to “client”.
5. Scroll up and select **Save**.

5.4.12.4.2 Ingress: Limit pods that access a database to this namespace

Suppose that while deploying an application in a project, you want to protect its database pods by permitting ingress only from API service pods in the current namespace, and prevent ingress from pods in any other namespace.

You can limit the database pods to just the incoming traffic from the current namespaces by adding an ingress rule with these characteristics:

- Use Port 3306 to receive incoming TCP traffic for pods that have the label, `tier: database`
- Refuse traffic from pods unless they have the label, `tier: api`

5.4.12.4.2.1 Configure the general information to access a database

1. Select the **+ Create Network Policy** button.
2. Type “database-access-api-only” in the **ID name** field.
3. Type “Allow MySQL access only from API pods in this namespace” in the **Description** field.
4. Select **Pod Selector** then select **Match Label**.
5. Set the **Key** to “tier” and the **Value** to “database”.

5.4.12.4.2.2 Create an Ingress rule to access a database

1. Select Default for the **Policy Types**.
2. Scroll down to the **Ingress** section.
3. Select **+ Add Ingress Rule**, then select **+ Add Port**.
4. Set the **Port** to “3306” and leave the **Protocol** set to “TCP”.

5.4.12.4.2.3 Add Sources to access a database

1. Select **+ Add Source**.
2. Select **+ Add Pod Selector**.
3. Select **Match Label** and set the **Key** to “tier” and the **Value** to “API”.
4. Scroll up and select **Save**.

5.4.12.4.3 Ingress: Disable but don’t delete ingress rules

Suppose that you want to disable ingress rules temporarily for testing or triaging purposes.

First, you need to create a network policy with one or more ingress rules. You could follow one of the preceding procedures, for example. Then, you need to edit the policy to match the following example:

5.4.12.4.3.1 Edit your Network Policy

1. In the table row belonging to your network policy, click the context menu at the right of the row and select **Edit**.

5.4.12.4.3.2 Disable Ingress rules

1. Update the **Policy Types** so that only **Egress** is selected. If you don’t want to deny *all* egress traffic, ensure that you add an egress rule that suits your preferred level of access. You can add an empty rule to allow all egress traffic.

2. Scroll up and select **Save**.

5.4.12.4.4 Egress: Deny all egress traffic from restricted pods

Suppose that you need to deny all egress traffic from a group of restricted pods. This is a simple egress rule and you can create it following this example and steps:

5.4.12.4.4.1 Configure the General Information to deny Egress

1. Select **+ Add Network Policy**.
2. Type “deny-restricted-egress” in the **ID name** field.
3. Type “Deny egress traffic from restricted pods” in the **Description** field.
4. Select **Pod selector** then select **Match Label.**
5. Set the **Key** to “access” and the **Value** to “restricted”.

5.4.12.4.4.2 Deny Egress traffic

1. Update the **Policy Types** so that only **Egress** is selected. Do not add any egress rules.
2. Scroll up and select **Save**.

5.5 Managing Clusters

View clusters created with Kommander or any connected Kubernetes cluster

Kommander allows you to monitor and manage very large numbers of clusters. Use the features described in this area to connect existing clusters, or to create new clusters whose lifecycle is managed by Konvoy. You can view clusters from the Clusters tab in the navigation pane on the left. You can see the details for a cluster by selecting the **View Details** link at the bottom of the cluster card or the cluster name in either the card or the table view.

- Cluster Applications and Statuses (see page 901)
- Kubernetes Cluster Federation (KubeFed) (see page 903)

5.5.1 Create a Managed Azure Cluster from the DKP UI

Enterprise Gov Advanced

DKP UI allows you to provision an Azure cluster from your browser.

5.5.1.1 Prerequisites

> Before you provision an Azure cluster using the DKP UI, you must first create an Azure infrastructure provider (see page 639) to hold your Azure credentials.

5.5.1.2 Provision an Azure Cluster

Follow these steps to provision an Azure cluster:

1. From the top menu bar, select your target workspace.
2. Select **Clusters > Add Cluster**.
3. Choose **Create Cluster**.
4. Enter the **Cluster Name**.
5. From **Select Infrastructure Provider**, choose the provider created in the prerequisites section.
6. If available, choose a **Kubernetes Version**. Otherwise, the DKP 2.7.0 Supported Kubernetes Versions (see page 1948) installs.
7. Select a data center **location** or specify a custom **location**.
8. Edit your worker **Node Pools** as necessary. You can choose the **Number of Nodes**, the **Machine Type**, and for the worker nodes you can choose a **Worker Availability Zone**.
9. Add any additional **Labels** or **Infrastructure Provider Tags** as necessary.
10. Review your inputs to ensure they meet the predefined criteria, and select **Create**.

> It can take up to 15 minutes for your cluster to appear in the **Provisioned** status.

You are then redirected to the **Clusters** page, where you'll see your new cluster in the **Provisioning** status. Hover over the status to view the details.

5.5.2 Create a Managed Cluster on vSphere from the DKP UI

Enterprise Gov Advanced

You can use the DKP user interface to provision a vSphere cluster quickly and easily.

5.5.2.1 Prerequisites

Ensure that you have fulfilled the vSphere vCenter configuration prerequisites described in vSphere: VMware Prerequisites (see page 446) before you begin these procedures.

> You must also create a vSphere infrastructure provider (see page 640) before you can create additional vSphere clusters.

5.5.2.2 Provision a vSphere Cluster

Provisioning a production-ready cluster in vSphere requires you to specify a fairly large number of parameters. Breaking up the sections of the form, as done below, makes it a little easier to complete.

Complete these procedures to provision a vSphere cluster:

5.5.2.2.1 Provide Basic Cluster Information

In the section of the provisioning form, you give the cluster a name and provide some basic information:

1. In the selected workspace Dashboard, select the **Add Cluster** button at the top right to display the Add Cluster page.
2. Select the **Create Cluster** card.
3. Provide these cluster details in the form:
   - **Cluster Name:** A valid Kubernetes name for the cluster.
   - **Add Labels**: Add any required Labels the cluster needs for your environment by selecting the `+ Add Label` link.
     By default, your cluster has labels that reflect the infrastructure provider provisioning. For example, your vSphere cluster may have a label for the data center and `provider: vsphere`. Cluster labels are matched to the selectors created for Projects (see page 716). Changing a cluster label may add or remove the cluster from projects.
   - **Infrastructure Provider**: This field's value corresponds to the vSphere infrastructure provider you created while fulfilling the prerequisites.
   - **Kubernetes Version**: Select a supported version of Kubernetes for this version of DKP.

- **SSH Public Key**: Paste into this field the public key value for a user who is authorized to create vSphere clusters.
- **Workspace**: The workspace where this cluster belongs (if within the Global workspace).

5.5.2.2.2 Specify the Cluster Resources and Network Information

This section of the form identifies already existing resources in your VMware vCenter configuration. Refer to your vCenter configuration to find the necessary values.

1. Provide the following values for the **Resources** that are specific to vSphere:
    - **Datacenter**: Select an existing data center name.
      The data-center is the top level organizational unit in vSphere.
    - **Datastore**: Enter a valid vSphere datastore name.
      Datastores in vSphere are storage resources that provide storage infrastructure for virtual machines within a data center. They are a subset of data center resources, with each datastore being associated with a specific data center.
    - **Folder**: Enter a valid, existing folder name, or leave it blank to use the vSphere root folder.
      When provisioning a Kubernetes cluster on vSphere using Cluster API and `clusterctl`, vSphere uses the folder parameter to specify the vSphere folder where it creates and manages the virtual machines for the Kubernetes cluster. Specifying the folder helps maintain an organized inventory of your virtual machines and other resources in your vSphere environment.
2. Enter the values for the network information in the lower half of this section:
    - **Network**: Enter an existing network name you want the new cluster to use.
      You need to create required network resources, such as port groups or distributed port groups, in the vSphere Client or using the vSphere API before you use DKP to create a new cluster.
    - **Resource Pool**: Enter the name of a logical resource pool for the cluster's resources.
      In vSphere, resource pools are a logical abstraction that allow you to allocate and manage computing resources, such as CPU and memory, for a group of virtual machines. Use resource pools only when needed, as they can add complexity to your environment.
    - **Virtual Machine Template**: Enter the name of the virtual machine template to use for the managed cluster's virtual machines.
      In vSphere, a virtual machine (VM) template is a pre-configured virtual machine that you can use to create new virtual machines with identical configurations quickly. The template contains the basic configuration settings for the VM, such as the operating system, installed software, and hardware configurations.
    - **Storage Policy**: Enter the name of a valid vSphere storage policy. This field is optional.
      A storage policy in vSphere specifies the storage requirements for virtual machine disks and files. It consists of a rule-set that defines the storage capabilities required, tags to identify it, profiles that collect settings and requirements, and storage requirements that include storage performance, capacity, redundancy, and other attributes necessary for the virtual machine to function properly. By creating and applying a storage policy to a specific datastore or group of

datastores, you can ensure that virtual machines using that datastore meet the specified storage requirements.

5.5.2.2.3 Configure Node Pool Information

You need to configure node pool information for both your control plane nodes and your worker nodes. The form splits these information sets into two groups.

1. Provide the control plane node pool name and resource sizing information:
    - **Node Pool Name**: DKP sets this field's value, `control-plane`, and you cannot change it.
    - **Disk**: Enter the amount of disk space allocated for each control plane node. The default value is 80 GB. The specified custom disk size must be equal to, or larger than the size of the base OS image root file system. This is because a root file system cannot be reduced automatically when a machine first boots.
    - **Memory**: The amount of memory for each control plane node, in GB. The default value is 16 GB.
    - **Number of CPUs**: Enter the number of virtual processors in each control plane node. The default value is 4 CPUs per control plane node.
    - **Replicas**: Enter the number of control plane nodes to create for your new cluster.
      Valid values for production clusters are 3 or 5. You can enter 1 if you are creating a test cluster, but a single control plane is not a valid production configuration. You must enter an odd number to allow for internal leader selection processes to provide proper failover for high availability. The default value is 3 control plane nodes.
2. Provide the worker node pool name and resource sizing information:
    - **Node Pool Name**: Enter a node pool name for the worker nodes. DKP sets this field's default value to `worker-0`.
    - **Disk**: Enter the amount of disk space allotted for each worker node. The default value is 80GB. The specified custom disk size must be equal to, or larger than the size of the base OS image root file system. This is because a root file system cannot be reduced automatically when a machine first boots.
    - **Memory**: The amount of memory for each worker node, in GB. The default value is 32 GB.
    - **Number of CPUs**: Enter the number of virtual processors in each worker node. The default value is 8 CPUs per node.
    - **Replicas**: Enter the number of worker nodes to create for your new cluster. The default value is 4 worker nodes.

5.5.2.2.4 Set Virtual IP Parameters

In this section of the form, you configure the built-in virtual IP.

1. Provide the Virtual IP information needed for managing this cluster with DKP:

- **Interface**: Enter the name of the network used for the virtual IP control plane endpoint.
  This value is specific to your environment and cannot be inferred by DKP. An example value is `eth0` or `ens5`.
- **Host**: Enter the control plane endpoint address.
  To use an external load balancer, set this value to the load balancer's IP address or hostname.
  To use the built-in virtual IP, set to a static IPv4 address in the Layer 2 network of the control plane machines.
- **Port**: Enter the control plane's endpoint port.
  The default port value is 6443. To use an external load balancer, se this value to the load's balancer's listening port.

5.5.2.2.5 Supply MetalLB Information

The MetalLB load balancer is needed for cluster installation, and requires these values:

1. Provide a **Starting IP** address range value for the load balancing allocation.
2. Provide an **Ending IP** address range value for the load balancing allocation.

5.5.2.2.6 Configure the StorageClass Options

In this section of the form, you configure the storage options for your vSphere cluster. The StorageClass defines the provisioning properties and requirements for the storage used to store the persistent data of the Kubernetes application.

> You can provide either the Datastore URL or the Storage Policy Name in this section.

- Select the **Datastore URL** button if it is not already highlighted, and then in the **Datastore URL** field, enter a unique identifier in URL format used by vSphere to access specific storage locations. A typical example of the field's format is `ds:///vmfs/volumes/<datastore_uuid>/`.
- Select the **Storage Policy Name** button if it is not already highlighted, and then in the **Storage Policy Name** field, enter the name of the storage policy to use with the cluster's StorageClass.

5.5.2.3 Advanced Configuration Parameters

You can open the Advanced configuration parameters sections by selecting the **Show Advanced** link.

5.5.2.3.1 Configure CIDR Values for the Pod Network and Kubernetes Services

In this section of the form, you configure Classless Inter-Domain Routing (CIDR) Values that your vSphere cluster uses.

1. Enter a CIDR value for the Pod network in the **Pod Network CIDR** field. The default value is 192.168.0.0/16.
2. Enter a CIDR value for Kubernetes Services in the **Service CIDR** field. The default value is 10.96.0.0/12.

5.5.2.3.2 Configure the Docker Registry Mirror

In this section, you configure a registry mirror for container images. The first time you request an image from your local registry mirror, it pulls the image from a public registry and stores it locally before handing it back to you. On subsequent requests, the local registry mirror serves the image from its own storage.

1. Configure the image registry mirror:
    - **Registry Mirror URL**: Enter the URL of a container registry to use as a registry mirror.
    - **Registry Mirror Username**: Enter the name of a user who can authenticate to the registry mirror.
    - **Registry Mirror Password**: Enter the password for the username in the previous entry.
    - **Registry Mirror CA Cert**: Upload a certificate file, or copy the CA certificate chain value into the provided field to use while communicating with the registry mirror using Transport Layer Security (TLS).
      This value is a trusted root certificate (or chain of certificates) that validates the SSL/TLS connection between clients and the registry mirror, ensuring secure and trustworthy communications.

5.5.2.3.3 Create the Managed Cluster on vSphere

Select the **Create** button (at the page's top right corner) to begin provisioning the cluster.

This step may take a few minutes, taking time for the cluster to be ready and fully deploy its components. The cluster automatically tries to join the management cluster for federation and fleet operations, and should resolve after it is fully provisioned.

While DKP provisions the new cluster, you can access the **Clusters** page to view the new cluster. A new cluster card with the name of your cluster appears, and shows a "Pending" cluster status while the cluster comes up and joins the management cluster.

5.5.2.4 Next Actions

Select the **View Details** link (on the cluster card's bottom left corner) to see additional information about this cluster.

5.5.3 Create a Managed Cluster on VCD Using the DKP UI

After configuring VMware Cloud Director (VCD), you can use the DKP user interface to provision a VCD cluster quickly and easily.

5.5.3.1 Prerequisites

Ensure that you have fulfilled the VMware Cloud Director configuration prerequisites described in the section, Cloud Director for Service Providers (see page 1445), before you begin these procedures.

> You must also create a VCD infrastructure provider (see page 640)before you can create additional VCD clusters.

5.5.3.2 Provision a VCD Cluster

Provisioning a production-ready cluster in VCD requires you to specify a fairly large number of parameters. Breaking up the sections of the form, as done below, makes it a little easier to complete.

Complete these procedures to provision a VCD cluster:

- Provide basic cluster information
- Specify an SSH user and public key
- Define the cluster's VCD resources
- Configure node pools for the cluster
- Specify pod and service CIDR values, if needed
- Define a registry mirror, if needed

5.5.3.2.1 Provide Basic Cluster Information

In this section of the provisioning form, you give the cluster a name and provide some basic information:

- **Cluster Name:** A valid Kubernetes name for the cluster.
- **Add Labels**: Add any required Labels the cluster needs for your environment by selecting the `+ Add Label` link. Changing a cluster Label may add or remove the cluster from Projects.
- **Infrastructure Provider**: This field's value corresponds to the VCD infrastructure provider you created while fulfilling the prerequisites.
- **Kubernetes Version**: Select a supported version of Kubernetes for this version of DKP.

5.5.3.2.2 Specify an SSH User

In this section, you specify the user name and public key values for an SSH user who has access to the VCD cluster after its creation. This group of credentials enables password-less SSH access for administrative tasks and debugging purposes, which improves security. DKP adds the public key to the `authorized_keys` file on each node, allowing the corresponding private key holder to access the nodes using SSH.

1. Type an **SSH Username** to which the SSH Public Key belongs.
   Leaving this field blank means that DKP assigns the user name `konvoy` to the public key value you copy into the next field.
2. Copy and paste an entire, valid **SSH Public Key** to go with the SSH Username in the previous field.

5.5.3.2.3 Define the Cluster's VCD Resources

This section of the form identifies already existing resources in your VMware Cloud Director configuration. Refer to your VCD configuration to find the necessary values.

1. Provide the following values for the **Resources** that are specific to VCD:
   - **Datacenter**: Select an existing Organization's Virtual Datacenter(VDC) name where you want to deploy the cluster.
   - **Network:** Select the Organization's virtual datacenter Network that the new cluster uses.
   - **Organization:** The Organization (see page 1452) name under which you want to deploy the cluster.
   - **Catalog:** The name of the VCD Catalog that hosts the virtual machine templates used for cluster creation.
   - **vApp Template:** The vApp template used to create the virtual machines that comprise the cluster. A DKP cluster becomes a vApp in VCD.
     Storage Profile: The name of a VCD storage profile you want to use for a virtual machine. The default value in DKP is " * " and selects the policy defined as the default storage policy in VCD.

5.5.3.2.4 Configure Node Pools

You need to configure node pool information for both your control plane nodes and your worker nodes. The form splits these information sets into two groups.

1. Provide the control plane node pool name and resource sizing information:
   - **Node Pool Name**: DKP sets this field's value, `control-plane`, and you cannot change it.
   - **Number of Nodes**: Enter the number of control plane nodes to create for your new cluster. Valid values for production clusters are 3 or 5. You can enter 1 if you are creating a test cluster, but a single control plane is not a valid production configuration. You must enter an odd number to allow for internal leader selection processes to provide proper failover for high availability. The default value is 3 control plane nodes.
   - **Placement Policy:** The placement policy for control planes to be used on this machine.
     A VM placement policy defines the placement of a virtual machine on a host or group of hosts. It is a mechanism for cloud provider administrators to create a named group of hosts within a provider VDC. The named group of hosts is a subset of hosts within the provider VDC clusters that might be selected based on any criteria such as performance tiers or licensing. You can expand the scope of a VM placement policy to more than one provider VDC.
   - **Sizing Policy:** The sizing policy for control planes to be used on this machine.
     A VM sizing policy defines the compute resource allocation for virtual machines within an

organization VDC. The compute resource allocation includes CPU and memory allocation, reservations, limits, and shares.

2. Provide the worker node pool name and resource sizing information:
    - **Node Pool Name**: Enter a node pool name for the worker nodes. DKP sets this field's default value to `worker-0`.
    - **Replicas**: Enter the number of worker nodes to create for your new cluster. The default value is 4 worker nodes.
    - **Placement Policy:** The placement policy for workers to be used on this machine.
      A VM placement policy defines the placement of a virtual machine on a host or group of hosts. It is a mechanism for cloud provider administrators to create a named group of hosts within a provider VDC. The named group of hosts is a subset of hosts within the provider VDC clusters that might be selected based on any criteria such as performance tiers or licensing. You can expand the scope of a VM placement policy to more than one provider VDC.
    - **Sizing Policy:** The sizing policy for workers to be used on this machine.
      A VM sizing policy defines the compute resource allocation for virtual machines within an organization VDC. The compute resource allocation includes CPU and memory allocation, reservations, limits, and shares.

5.5.3.3 Advanced Configuration

5.5.3.3.1 Specify CIDR Values (group)

In this section, you specify the CIDR blocks for Pods and Services in your VCD cluster. You also need to reserve a separate CIDR block for Services. These ranges must not overlap each other, or the existing network. Incorrect configuration can cause network conflicts and disrupt cluster operations.

1. Specify the **Pod Network CIDR** to use in VCD clusters.
   The default value is 192.168.0.0/16.
2. Specify the **Service CIDR** to use in VCD clusters.
   The default value is 10.96.0.0/12.

5.5.3.3.2 Define a Registry Mirror (group)

A registry mirror (see page 1606) is a local caching proxy for a container image repository. When clients request an image, the mirror first tries to serve the image from its cache. If the image is not available in the cache, the mirror fetches it from the primary repository, caches it, and then serves it to the client.

1. Enter the **URL** of a container registry to use as a mirror in the cluster.
2. Type the **Username** for the account to use to authenticate to the registry mirror.
3. Type the **Password** for the account to authenticate to the registry mirror.
4. Copy and paste a **CA Certificate** chain to use while communicating with the registry mirror using TLS.

5.5.3.3.3 Control Plane Endpoint

In this section, you specify the control plane endpoint host and port values that enables both IP addresses and DNS names to map to the IP address for this cluster.

1. Type the **Host** name as either the control plane endpoint IP or a hostname.
2. Enter a **Port** value for the control plane endpoint port. The default value is 6443. To use an external load balancer, set this port value to to the load balancer's listening port number.

5.5.4 Attach a Kubernetes Cluster

Enterprise Gov Advanced

Attach an existing Kubernetes cluster using kubeconfig

5.5.4.1 Attach Kubernetes Cluster

You can attach an existing cluster (whether it is a cluster created with the DKP CLI or by means of another platform) to DKP. At the time of attachment, certain namespaces are created on the cluster, and workspace platform applications are deployed automatically into the newly-created namespaces.

Review the Workspace Platform Application Configuration Requirements (see page 124) to ensure the attached cluster has sufficient resources. For more information on platform applications and customizing them, see Workspace Applications (see page 679).

If the cluster you want to attach was created using Amazon EKS, Azure AKS or Google GKE, create a service account as described in Attach a Cluster with no Networking Restrictions (UI) (see page 790).

> Starting with DKP 2.6.0, DKP supports the attachment of all Kubernetes Conformant clusters, but only x86-64 architecture is supported, not ARM.

Platform applications extend the functionality of Kubernetes and provide ready-to-use logging and monitoring stacks by deploying platform applications when attaching a cluster to Kommander. For more information, refer to Workspace Applications (see page 679).

- Requirements for Attaching an Existing Cluster (see page 785)
- Create a kubeconfig File for Attachment (see page 787)
- Attach a Cluster with no Networking Restrictions (UI) (see page 790)
- Attach a Cluster with Networking Restrictions (see page 804)
- Attach a DKP-created Kubernetes Cluster (see page 856)
- Access a Managed or Attached Cluster (see page 858)
- Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859)

5.5.4.2 Requirements for Attaching an Existing Cluster

Enterprise Gov Advanced

5.5.4.2.1 Basic Requirements

To attach an existing cluster in the UI, the Application Management cluster must be able to reach the services and the `api-server` of the target cluster.

The cluster you want to attach can be a **DKP-CLI-created** cluster (which will become a Managed cluster (see page 80) upon attachment), or **another Kubernetes cluster** like AKS, EKS, or GKE (which will become an Attached cluster (see page 80) upon attachment).

> Starting with DKP 2.6.0, DKP supports the attachment of all Kubernetes Conformant clusters, but only x86-64 architecture is supported, not ARM.

For attaching existing clusters without networking restrictions, the requirements depend on which DKP version you are using. Each version of DKP supports a specific range of Kubernetes versions (see page 1948). You must ensure that the target cluster is running a compatible version.

> As of this version of DKP, there is no official method of uninstalling Kommander.

5.5.4.2.2 Create a Default StorageClass

To deploy many of the services on the attached cluster, there must be a default `StorageClass` configured. Run the following command on the cluster you want to attach:

```
kubectl get sc
```

The output should look similar to this. Note the `(default)` after the name:

```
NAME              PROVISIONER       RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
ebs-sc (default)   ebs.csi.aws.com   Delete          WaitForFirstConsumer   false
41s
```

If the `StorageClass` is not set as default, add the following annotation to the `StorageClass` manifest:

```
annotations:
  storageclass.kubernetes.io/is-default-class: "true"
```

5.5.4.2.3 Creating Projects and Workspaces

Before you attach clusters, you need to create one or more Workspaces, and we recommend that you also create Projects within your Workspaces. Workspaces (see page 678) give you a logical way to represent your teams and specific configurations. Projects (see page 716) let you define one or more clusters as a group to which Kommander pushes a common configuration. Grouping your existing clusters in Kommander projects and workspaces makes managing their platform services and resources easier and supports monitoring and logging.

> Do not attach a cluster in the "Management Cluster Workspace" workspace. This workspace is reserved for your Application Management cluster only.

5.5.4.2.4 Platform Application Requirements

In addition to the basic cluster requirements, the platform services you want DKP to manage on those clusters will have an impact on the total cluster requirements. The specific combinations of platform applications will make a difference in the requirements for the cluster nodes and their resources (CPU, memory, and storage).

See this table of platform applications (see page 662) that DKP provides by default.

5.5.4.2.5 Attach Existing AKS, EKS and GKE clusters

Attaching an existing AWS cluster requires that the cluster be fully configured and running (see page 1190). You must create a separate service account when attaching existing AKS, EKS or Google GKE Kubernetes clusters. This is necessary because the kubeconfig files generated from those clusters are not usable out-of-the-box by Kommander. The kubeconfig files call CLI commands, such as `azure, aws` or `gcloud`, and use locally-obtained authentication tokens. Having a separate service account also allows you to keep access to the cluster specific and isolated to Kommander.

The suggested default cluster configuration includes a control plane pool containing three (3) m5.xlarge nodes and a worker pool containing four (4) m5.2xlarge nodes.

Consider the additional resource requirements for running the platform services you want DKP to manage, and ensure that your existing clusters comply.

To attach an existing EKS cluster, refer to the specific information in Attach Amazon EKS Cluster (see page 1278).

To attach an existing GKE cluster, refer to the specific information in Attach GKE Cluster (see page 800).

5.5.4.2.6 Attach Clusters with an Existing cert-manager Installation

If you are attaching clusters that already have cert-manager installed, the cert-manager `HelmRelease` provided by DKP will fail to deploy, due to the existing cert-manager installation. As long as the pre-existing cert-manager functions as expected, you can ignore this failure. It will have no impact on the operation of the cluster.

5.5.4.3 Create a kubeconfig File for Attachment

Create a separate service account when attaching existing clusters (for example Amazon EKS, or Azure AKS clusters)

> If you already have a `kubeconfig` file to attach your cluster, go directly to Attach a Cluster with no Networking Restrictions (UI) (see page 790) or Attach a Cluster with Networking Restrictions (see page 804).

The `kubeconfig` files generated from existing clusters are not usable out-of-the-box, because they call provisioner-specific CLI commands (like `aws` commands), and use locally-obtained authentication tokens that are not compatible with DKP. Having a separate service account also allows you to have a dedicated identity for all DKP operations.

To get started, ensure you have kubectl[569] set up and configured with ClusterAdmin[570] for the cluster you want to connect to Kommander.

1. Create the necessary service account:

```
kubectl -n kube-system create serviceaccount kommander-cluster-admin
```

2. Create a token secret for the `serviceaccount`:

```
kubectl -n kube-system create  -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: kommander-cluster-admin-sa-token
  annotations:
    kubernetes.io/service-account.name: kommander-cluster-admin
type: kubernetes.io/service-account-token
EOF
```

569 https://kubernetes.io/docs/tasks/tools/#kubectl
570 https://kubernetes.io/docs/concepts/cluster-administration/

3. Verify that the `serviceaccount` token is ready by running this command:

```
kubectl -n kube-system get secret kommander-cluster-admin-sa-token -oyaml
```

   Verify that the `data.token` field is populated. The output should be similar to this:

```
apiVersion: v1
data:
  ca.crt: LS0tLS1CRUdJTiBDR...
  namespace: ZGVmYXVsdA==
  token: ZXlKaGJHY2lPaUpTVX...
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: kommander-cluster-admin
    kubernetes.io/service-account.uid: b62bc32e-b502-4654-921d-94a742e273a8
  creationTimestamp: "2022-08-19T13:36:42Z"
  name: kommander-cluster-admin-sa-token
  namespace: default
  resourceVersion: "8554"
  uid: 72c2a4f0-636d-4a70-9f1c-55a75f15e520
type: kubernetes.io/service-account-token
```

4. Configure the new service account for `cluster-admin` permissions:

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kommander-cluster-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kommander-cluster-admin
  namespace: kube-system
EOF
```

5. Set up the following environment variables with the access data that is needed for producing a new kubeconfig file:

```
export USER_TOKEN_VALUE=$(kubectl -n kube-system get secret/kommander-cluster-
admin-sa-token -o=go-template='{{.data.token}}' | base64 --decode)
export CURRENT_CONTEXT=$(kubectl config current-context)
```

```
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-
template='{{range .contexts}}{{if eq .name "'''${CURRENT_CONTEXT}'''"}}
{{ index .context "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}"{{with
index .cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{ end }}')
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}
{{ .cluster.server }}{{end}}{{ end }}')
```

6. Confirm these variables have been set correctly:

```
export -p USER_TOKEN_VALUE CURRENT_CONTEXT CURRENT_CLUSTER CLUSTER_CA
CLUSTER_SERVER
```

7. Generate a `kubeconfig` file that uses the environment variable values from the previous step:

```
cat << EOF > kommander-cluster-admin-config
apiVersion: v1
kind: Config
current-context: ${CURRENT_CONTEXT}
contexts:
- name: ${CURRENT_CONTEXT}
  context:
    cluster: ${CURRENT_CONTEXT}
    user: kommander-cluster-admin
    namespace: kube-system
clusters:
- name: ${CURRENT_CONTEXT}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
users:
- name: kommander-cluster-admin
  user:
    token: ${USER_TOKEN_VALUE}
EOF
```

8. This process produces a file in your current working directory called `kommander-cluster-admin-config`. The contents of this file are used in Kommander to attach the cluster.
   Before importing this configuration, verify the `kubeconfig` file can access the cluster:

```
kubectl --kubeconfig $(pwd)/kommander-cluster-admin-config get all --all-
namespaces
```

5.5.4.3.1 Next Step:

Use this `kubeconfig` to:

- Attach a cluster with no additional networking restrictions (see page 790)
- Attach a cluster that has networking restrictions (see page 804)

> If a cluster has limited resources to deploy all the federated platform services, it will fail to stay attached in the DKP UI. If this happens, check if there are any pods that are not getting the resources required.

5.5.4.4 Attach a Cluster with no Networking Restrictions (UI)

Enterprise Gov Advanced

For provider-specific instructions, refer to:

- Attach Amazon EKS Cluster (see page 791)
- Attach AKS Cluster (see page 795)
- Attach GKE Cluster (see page 800)

Using the **Add Cluster** option, you can attach an existing Kubernetes or DKP cluster directly to DKP. This enables you to access the multi-cluster management and monitoring benefits that DKP provides, while keeping your existing cluster on its current provider and infrastructure.

How to attach an existing cluster that has no additional networking restrictions

Use this option when you want to attach a cluster that does not require additional access information.

1. From the top menu bar, select your target workspace.
2. On the Dashboard page, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
3. Select **Attach Cluster**.
4. Select the **No additional networking restrictions** card. Alternatively, if you MUST use network restrictions, stop following the steps below, and see the instructions on the page Attach a Cluster with Network Restrictions (see page 804).
5. In the **Cluster Configuration** section, paste your `kubeconfig` file into the field, or select the **Upload kubeconfig File** button to specify the file.
6. The **Cluster Name** field will automatically populate with the name of the cluster is in the `kubeconfig`. You can edit this field with the name you want for your cluster.

7. The **Context** select list is populated from the `kubeconfig`. Select the desired context with admin privileges from the **Context** select list.
8. Add labels to classify your cluster as needed.
9. Select **Create** to attach your cluster.

5.5.4.4.1 Attach Amazon EKS Cluster

Enterprise Gov Advanced

5.5.4.4.1.1 Attach an existing EKS cluster

You can attach existing Kubernetes clusters to the Management Cluster. After attaching the cluster, you can use the UI to examine and manage (see page 774) this cluster. The following procedure shows how to attach an existing Amazon Elastic Kubernetes Service (EKS) cluster.

5.5.4.4.1.2 Before you Begin

This procedure requires the following items and configurations:

- A fully configured and running Amazon EKS[571] cluster with administrative privileges.
- The current version DKP Enterprise is installed (see page 1530) on your cluster.
- Ensure you have installed `kubectl` in your Management cluster.

This procedure assumes you have an existing and spun up Amazon EKS cluster(s) with administrative privileges. Refer to the Amazon EKS[572] for setup and configuration information.

5.5.4.4.1.3 Attach Amazon EKS Clusters

Ensure that the `KUBECONFIG` environment variable is set to the Management cluster before attaching by running:

```
export KUBECONFIG=<Management_cluster_kubeconfig>.conf
```

Access your EKS clusters

571 https://aws.amazon.com/eks/
572 https://aws.amazon.com/eks/

1. Ensure you are connected to your EKS clusters. Enter the following commands for each of your clusters:

```
kubectl config get-contexts
kubectl config use-context <context for first eks cluster>
```

2. Confirm `kubectl` can access the EKS cluster:

```
kubectl get nodes
```

Create a `kubeconfig` File for your EKS cluster

To get started, ensure you have kubectl[573] set up and configured with ClusterAdmin[574] for the cluster you want to connect to Kommander.

1. Create the necessary service account:

```
kubectl -n kube-system create serviceaccount kommander-cluster-admin
```

2. Create a token secret for the `serviceaccount`:

```
kubectl -n kube-system create  -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: kommander-cluster-admin-sa-token
  annotations:
    kubernetes.io/service-account.name: kommander-cluster-admin
type: kubernetes.io/service-account-token
EOF
```

For more information on Service Account Tokens, refer to this article[575] in our blog.

3. Verify that the `serviceaccount` token is ready by running this command:

```
kubectl -n kube-system get secret kommander-cluster-admin-sa-token -oyaml
```

Verify that the `data.token` field is populated. The output should be similar to this:

573 https://kubernetes.io/docs/tasks/tools/#kubectl
574 https://kubernetes.io/docs/concepts/cluster-administration/
575 https://eng.d2iq.com/blog/service-account-tokens-in-kubernetes-v1.24/#whats-changed-in-kubernetes-v124

```
apiVersion: v1
data:
  ca.crt: LS0tLS1CRUdJTiBDR...
  namespace: ZGVmYXVsdA==
  token: ZXlKaGJHY2lPaUpTVX...
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: kommander-cluster-admin
    kubernetes.io/service-account.uid: b62bc32e-b502-4654-921d-94a742e273a8
  creationTimestamp: "2022-08-19T13:36:42Z"
  name: kommander-cluster-admin-sa-token
  namespace: default
  resourceVersion: "8554"
  uid: 72c2a4f0-636d-4a70-9f1c-55a75f15e520
type: kubernetes.io/service-account-token
```

4. Configure the new service account for `cluster-admin` permissions:

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kommander-cluster-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kommander-cluster-admin
  namespace: kube-system
EOF
```

5. Set up the following environment variables with the access data that is needed for producing a new kubeconfig file:

```
export USER_TOKEN_VALUE=$(kubectl -n kube-system get secret/kommander-cluster-
admin-sa-token -o=go-template='{{.data.token}}' | base64 --decode)
export CURRENT_CONTEXT=$(kubectl config current-context)
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-
template='{{range .contexts}}{{if eq .name "'''${CURRENT_CONTEXT}'''"}}
{{ index .context "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}"{{with
index .cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{ end }}')
```

```
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}
{{ .cluster.server }}{{end}}{{ end }}')
```

6. Confirm these variables have been set correctly:

```
export -p | grep -E 'USER_TOKEN_VALUE|CURRENT_CONTEXT|CURRENT_CLUSTER|
CLUSTER_CA|CLUSTER_SERVER'
```

7. Generate a kubeconfig file that uses the environment variable values from the previous step:

```
cat << EOF > kommander-cluster-admin-config
apiVersion: v1
kind: Config
current-context: ${CURRENT_CONTEXT}
contexts:
- name: ${CURRENT_CONTEXT}
  context:
    cluster: ${CURRENT_CONTEXT}
    user: kommander-cluster-admin
    namespace: kube-system
clusters:
- name: ${CURRENT_CONTEXT}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
users:
- name: kommander-cluster-admin
  user:
    token: ${USER_TOKEN_VALUE}
EOF
```

8. This process produces a file in your current working directory called `kommander-cluster-admin-config`. The contents of this file are used in Kommander to attach the cluster.
   Before importing this configuration, verify the `kubeconfig` file can access the cluster:

```
kubectl --kubeconfig $(pwd)/kommander-cluster-admin-config get all --all-
namespaces
```

Finalize Attachment of your Cluster from the UI

Now that you have kubeconfig, go to the DKP UI and follow these steps below:

1. From the top menu bar, select your target workspace.
2. On the Dashboard page, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.

3. Select **Attach Cluster**.
4. Select the **No additional networking restrictions** card. Alternatively, if you must use network restrictions, stop following the steps below, and see the instructions on the page Attach a cluster WITH network restrictions (see page 804).
5. Upload the kubeconfig file you created in the previous section (or copy its contents) into the **Cluster Configuration** section.
6. The **Cluster Name** field automatically populates with the name of the cluster in the kubeconfig. You can edit this field with the name you want for your cluster.
7. Add labels to classify your cluster as needed.
8. Select **Create** to attach your cluster.

If a cluster has limited resources to deploy all the federated platform services, it will fail to stay attached in the DKP UI. If this happens, ensure your system has sufficient resources for all pods.

5.5.4.4.1.4 Related Information

For information on related topics or procedures, refer to the following:

- Configuring and Running Amazon EKS Clusters[576]
- Installing and Configuring Konvoy v2.0 or above (see page 1259)
- Installing and Configuring Kommander v2.0 or above (see page 1538)
- Manage Clusters (see page 774)

5.5.4.4.2 Attach AKS Cluster

ENTERPRISE

5.5.4.4.2.1 Attach an existing AKS cluster

You can attach existing Kubernetes clusters to the Management Cluster. After attaching the cluster, you can use the UI to examine and manage (see page 774) this cluster. The following procedure shows how to attach an existing Azure Kubernetes Service (AKS) cluster.

5.5.4.4.2.2 Before you Begin

This procedure requires the following items and configurations:

- A fully configured and running Azure AKS[577]cluster with administrative privileges.

576 https://aws.amazon.com/eks/
577 https://azure.microsoft.com/en-us/products/kubernetes-service/

- The current version DKP Enterprise is installed (see page 146)on your cluster.
- Ensure you have installed `kubectl` in your Management cluster.

> This procedure assumes you have an existing and spun up Azure AKS cluster(s) with administrative privileges. Refer to the Azure AKS[578]for setup and configuration information.

5.5.4.4.2.3 Attach AKS Clusters

Ensure that the `KUBECONFIG` environment variable is set to the Management cluster before attaching by running:

```
export KUBECONFIG=<Management_cluster_kubeconfig>.conf
```

Ensure you have access to your AKS clusters

1. Ensure you are connected to your AKS clusters. Enter the following commands for each of your clusters:

   ```
   kubectl config get-contexts
   kubectl config use-context <context for first AKS cluster>
   ```

2. Confirm `kubectl` can access the AKS cluster:

   ```
   kubectl get nodes
   ```

Create a kubeconfig file for your AKS cluster

To get started, ensure you have kubectl[579] set up and configured with ClusterAdmin[580] for the cluster you want to connect to Kommander.

1. Create the necessary service account:

   ```
   kubectl -n kube-system create serviceaccount kommander-cluster-admin
   ```

578 https://azure.microsoft.com/en-us/products/kubernetes-service/
579 https://kubernetes.io/docs/tasks/tools/#kubectl
580 https://kubernetes.io/docs/concepts/cluster-administration/

2. Create a token secret for the `serviceaccount`:

```
kubectl -n kube-system create  -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: kommander-cluster-admin-sa-token
  annotations:
    kubernetes.io/service-account.name: kommander-cluster-admin
type: kubernetes.io/service-account-token
EOF
```

3. Verify that the `serviceaccount` token is ready by running this command:

```
kubectl -n kube-system get secret kommander-cluster-admin-sa-token -oyaml
```

Verify that the `data.token` field is populated. The output should be similar to this:

```
apiVersion: v1
data:
  ca.crt: LS0tLS1CRUdJTiBDR...
  namespace: ZGVmYXVsdA==
  token: ZXlKaGJHY2lPaUpTVX...
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: kommander-cluster-admin
    kubernetes.io/service-account.uid: b62bc32e-b502-4654-921d-94a742e273a8
  creationTimestamp: "2022-08-19T13:36:42Z"
  name: kommander-cluster-admin-sa-token
  namespace: default
  resourceVersion: "8554"
  uid: 72c2a4f0-636d-4a70-9f1c-55a75f15e520
type: kubernetes.io/service-account-token
```

4. Configure the new service account for `cluster-admin` permissions:

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kommander-cluster-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
```

```
subjects:
- kind: ServiceAccount
  name: kommander-cluster-admin
  namespace: kube-system
EOF
```

5. Set up the following environment variables with the access data that is needed for producing a new kubeconfig file:

```
export USER_TOKEN_VALUE=$(kubectl -n kube-system get secret/kommander-cluster-
admin-sa-token -o=go-template='{{.data.token}}' | base64 --decode)
export CURRENT_CONTEXT=$(kubectl config current-context)
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-
template='{{range .contexts}}{{if eq .name "'''${CURRENT_CONTEXT}'''"}}
{{ index .context "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}"{{with
index .cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{ end }}')
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}
{{ .cluster.server }}{{end}}{{ end }}')
```

6. Confirm these variables have been set correctly:

```
export -p | grep -E 'USER_TOKEN_VALUE|CURRENT_CONTEXT|CURRENT_CLUSTER|
CLUSTER_CA|CLUSTER_SERVER'
```

7. Generate a kubeconfig file that uses the environment variable values from the previous step:

```
cat << EOF > kommander-cluster-admin-config
apiVersion: v1
kind: Config
current-context: ${CURRENT_CONTEXT}
contexts:
- name: ${CURRENT_CONTEXT}
  context:
    cluster: ${CURRENT_CONTEXT}
    user: kommander-cluster-admin
    namespace: kube-system
clusters:
- name: ${CURRENT_CONTEXT}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
users:
- name: kommander-cluster-admin
  user:
    token: ${USER_TOKEN_VALUE}
```

```
EOF
```

8. This process produces a file in your current working directory called `kommander-cluster-admin-config`. The contents of this file are used in Kommander to attach the cluster.
   Before importing this configuration, verify the `kubeconfig` file can access the cluster:

   ```
   kubectl --kubeconfig $(pwd)/kommander-cluster-admin-config get all --all-namespaces
   ```

Finalize attaching your cluster from the UI

Now that you have kubeconfig, go to the DKP UI and follow these steps below:

1. From the top menu bar, select your target workspace.
2. On the Dashboard page, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
3. Select **Attach Cluster**.
4. Select the **No additional networking restrictions** card. Alternatively, if you must use network restrictions, stop following the steps below, and see the instructions on the page Attach a cluster WITH network restrictions (see page 804).
5. Upload the kubeconfig file you created in the previous section (or copy its contents) into the **Cluster Configuration** section.
6. The **Cluster Name** field automatically populates with the name of the cluster in the kubeconfig. You can edit this field with the name you want for your cluster.
7. Add labels to classify your cluster as needed.
8. Select **Create** to attach your cluster.

If a cluster has limited resources to deploy all the federated platform services, it will fail to stay attached in the DKP UI. If this happens, ensure your system has sufficient resources for all pods.

5.5.4.4.2.4 Related Information

For information on related topics or procedures, refer to the following:

- Installing and Configuring Konvoy v2.0 or above (see page 1343)
- Installing and Configuring Kommander v2.0 or above (see page 1532)
- Manage Clusters (see page 774)

5.5.4.4.3 Attach GKE Cluster

Enterprise Gov Advanced

Attach an existing GKE cluster in DKP

You can attach existing Kubernetes clusters to the Management Cluster. After attaching the cluster, you can use the UI to examine and manage (see page 774) this cluster. The following procedure shows how to attach an existing **standard** GKE cluster.

5.5.4.4.3.1 Before you Begin

This procedure requires the following items and configurations:

- A fully configured and running GKE cluster[581] with a supported Kubernetes version[582], and administrative privileges.
- The current version of DKP Enterprise installed (see page 146)on your cluster.
- Ensure you have installed `kubectl` in your Management cluster.

This procedure assumes you have an existing and spun up GKE cluster with administrator privileges.

5.5.4.4.3.2 Attach GKE Clusters

Ensure you have access to your GKE clusters

1. Ensure you are connected to your GKE clusters. Enter the following commands for each of your clusters:

```
kubectl config get-contexts
kubectl config use-context <context for first gcloud cluster>
```

2. Confirm `kubectl` can access the GKE cluster.

```
kubectl get nodes
```

581 https://cloud.google.com/kubernetes-engine/docs/concepts/types-of-clusters
582 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/150765792/DKP+2.5.0+Supported+Kubernetes+Versions

Configure a kubeconfig file

To get started, ensure you have kubectl[583] set up and configured with ClusterAdmin[584] for the cluster you want to connect to Kommander.

1. Create the necessary service account:

```
kubectl -n kube-system create serviceaccount kommander-cluster-admin
```

2. Create a token secret for the `serviceaccount`:

```
kubectl -n kube-system create  -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: kommander-cluster-admin-sa-token
  annotations:
    kubernetes.io/service-account.name: kommander-cluster-admin
type: kubernetes.io/service-account-token
EOF
```

   For more information on Service Account Tokens, refer to this article[585] in our blog.

3. Verify that the `serviceaccount` token is ready by running this command:

```
kubectl -n kube-system get secret kommander-cluster-admin-sa-token -oyaml
```

   Verify that the `data.token` field is populated. The output should be similar to this:

```
apiVersion: v1
data:
  ca.crt: LS0tLS1CRUdJTiBDR...
  namespace: ZGVmYXVsdA==
  token: ZXlKaGJHY2lPaUpTVX...
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: kommander-cluster-admin
    kubernetes.io/service-account.uid: b62bc32e-b502-4654-921d-94a742e273a8
  creationTimestamp: "2022-08-19T13:36:42Z"
  name: kommander-cluster-admin-sa-token
  namespace: default
```

583 https://kubernetes.io/docs/tasks/tools/#kubectl
584 https://kubernetes.io/docs/concepts/cluster-administration/
585 https://eng.d2iq.com/blog/service-account-tokens-in-kubernetes-v1.24/#whats-changed-in-kubernetes-v124

```
  resourceVersion: "8554"
  uid: 72c2a4f0-636d-4a70-9f1c-55a75f15e520
type: kubernetes.io/service-account-token
```

4. Configure the new service account for `cluster-admin` permissions:

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kommander-cluster-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kommander-cluster-admin
  namespace: kube-system
EOF
```

5. Set up the following environment variables with the access data that is needed for producing a new kubeconfig file:

```
export USER_TOKEN_VALUE=$(kubectl -n kube-system get secret/kommander-cluster-
admin-sa-token -o=go-template='{{.data.token}}' | base64 --decode)
export CURRENT_CONTEXT=$(kubectl config current-context)
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-
template='{{range .contexts}}{{if eq .name "'''${CURRENT_CONTEXT}'''"}}
{{ index .context "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}"{{with
index .cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{ end }}')
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}
{{ .cluster.server }}{{end}}{{ end }}')
```

6. Confirm these variables have been set correctly:

```
export -p | grep -E 'USER_TOKEN_VALUE|CURRENT_CONTEXT|CURRENT_CLUSTER|
CLUSTER_CA|CLUSTER_SERVER'
```

7. Generate a kubeconfig file that uses the environment variable values from the previous step:

```
cat << EOF > kommander-cluster-admin-config
apiVersion: v1
```

```
kind: Config
current-context: ${CURRENT_CONTEXT}
contexts:
- name: ${CURRENT_CONTEXT}
  context:
    cluster: ${CURRENT_CONTEXT}
    user: kommander-cluster-admin
    namespace: kube-system
clusters:
- name: ${CURRENT_CONTEXT}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
users:
- name: kommander-cluster-admin
  user:
    token: ${USER_TOKEN_VALUE}
EOF
```

8. This process produces a file in your current working directory called `kommander-cluster-admin-config`. The contents of this file are used in Kommander to attach the cluster.
   Before importing this configuration, verify the `kubeconfig` file can access the cluster:

   ```
   kubectl --kubeconfig $(pwd)/kommander-cluster-admin-config get all --all-namespaces
   ```

Attach the Cluster

Now that you have a kubeconfig file, go to the DKP UI and follow these steps:

1. Select your target workspace from the top menu bar.
2. Select **Add Cluster** in the **Actions** dropdown menu from the Dashboard page, located at the top-right.
3. Select **Attach Cluster**.
4. Select the **No additional networking restrictions** card.
   Alternatively, if you must use network restrictions, stop following the steps below, and see the instructions on the page Attach a cluster WITH network restrictions (see page 804).
5. Upload the kubeconfig file you created in the previous section (or copy its contents) into the **Cluster Configuration** section.
6. The **Cluster Name** field automatically populates with the name of the cluster in the kubeconfig. You can edit this field with the name you want for your cluster.
7. Add labels to classify your cluster as needed.
8. Select **Create** to attach your cluster.

If a cluster has limited resources to deploy all the federated platform services, it will fail to stay attached in the DKP UI. If this happens, ensure your system has sufficient resources for all pods.

5.5.4.4.3.3 Related Information

For information on related topics or procedures, refer to the following:

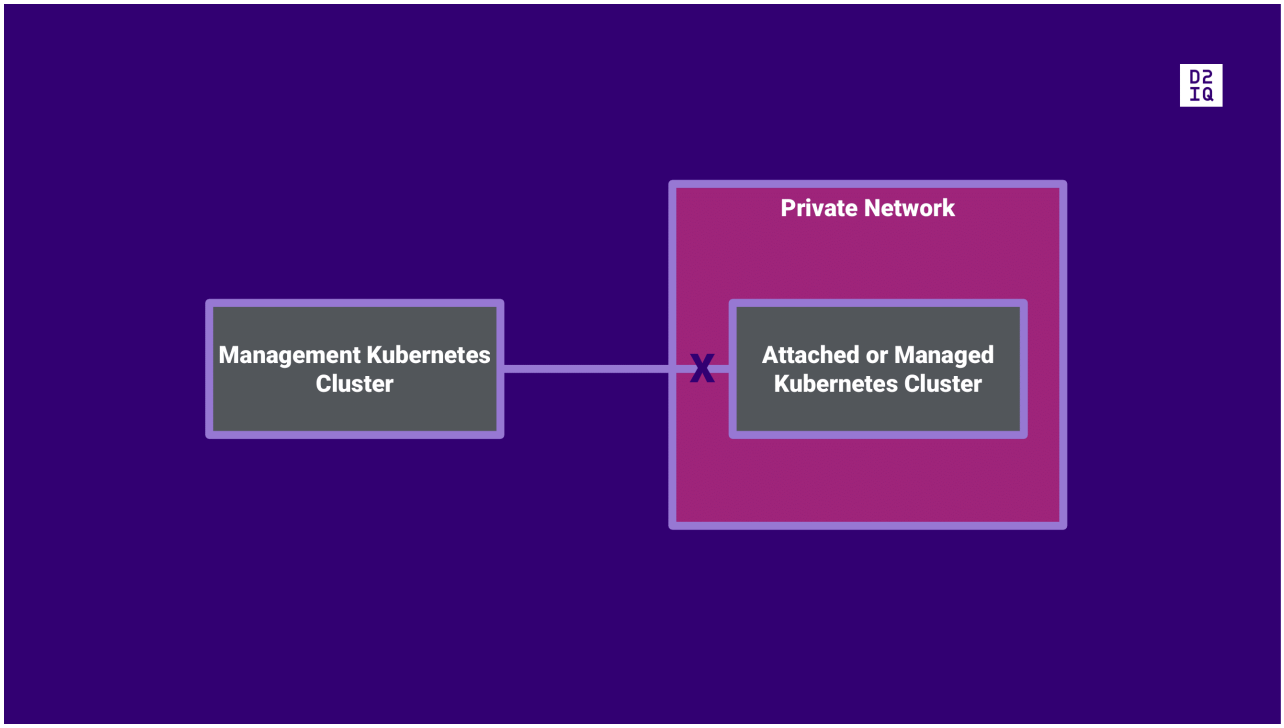
- Installing and Configuring Kommander (see page 1532)
- Manage Clusters (see page 774)

5.5.4.5 Attach a Cluster with Networking Restrictions

Enterprise Gov Advanced

5.5.4.5.1 When do I Need to use a Secure Tunnel?

When attaching a cluster to DKP, the Management cluster initiates an outbound connection to the cluster you wish to attach. This is not possible if the cluster you want to attach (Managed or Attached (see page 79)) has networking restrictions and is not exposed, for example, because it is in a private network, or its API is not accessible from the same network as the Management cluster. This is what we call a network-restricted cluster (see page 82).



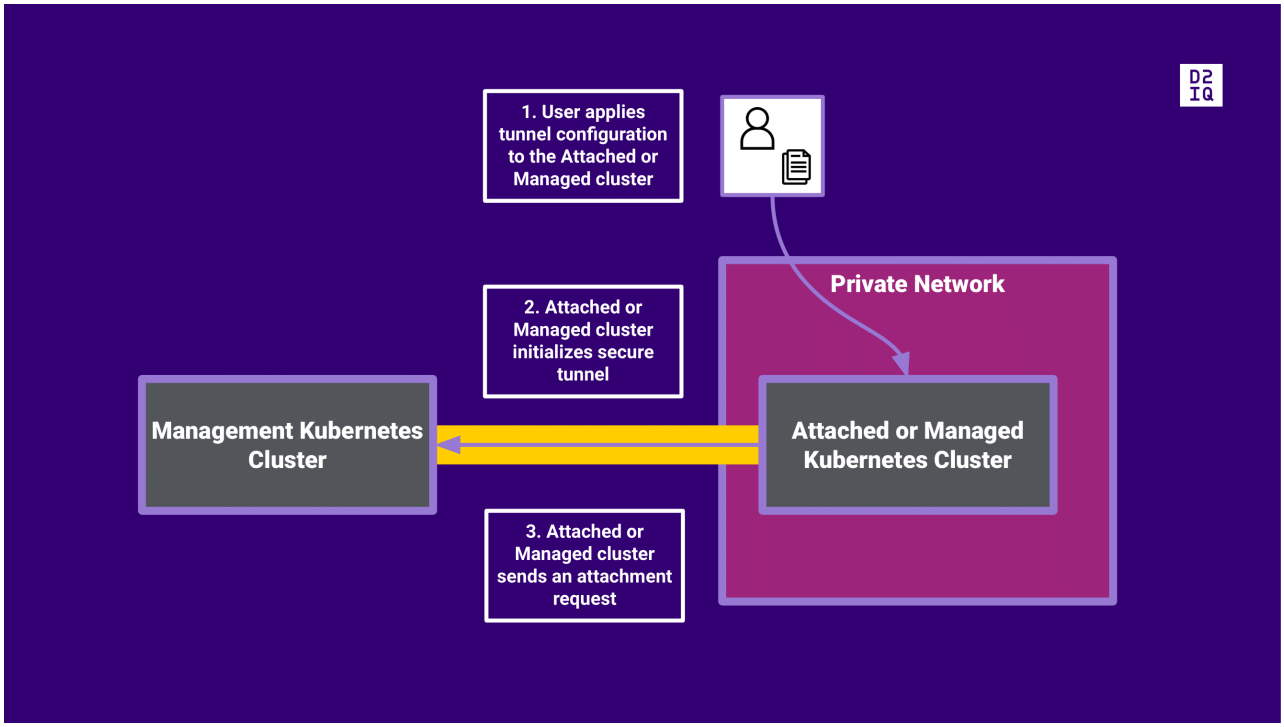
3 Diagram: Attachment not possible when cluster is in a private network.

Use a tunnel when you want to attach a cluster that is in a DMZ, behind a NAT gateway, behind a proxy server, firewall, or requires additional access information.

5.5.4.5.2 How Does the Tunneled Attachment work?

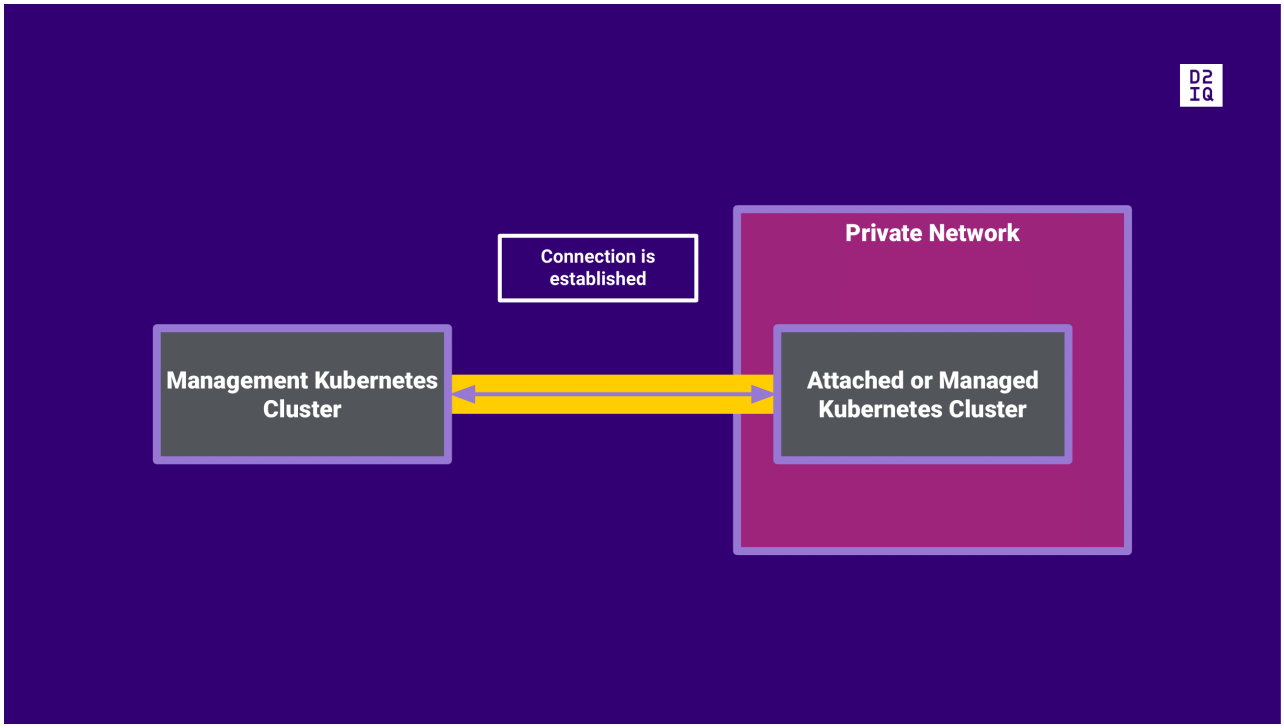
DKP can create a secure tunnel to enable the attachment of clusters that are not directly reachable.

To create a secure tunnel, you must provide a configuration for the tunnel in the cluster you want to attach. After you apply that configuration, the cluster you want to attach will establish a secure tunnel with the Management cluster, and make an attachment request.



4 Diagram: Attachment through a secure tunnel

After the attachment request is accepted and the connection between clusters is established, both clusters will allow bilateral communication.



5 Diagram: Connection between clusters is established

5.5.4.5.3 What are the Steps to Attach a Cluster via a Tunnel?

Ensure you meet the prerequisites for a tunneled attachment, and then attach the cluster with the UI or CLI:

- Prerequisites for a Tunneled Attachment (see page 806)
- Attach a Cluster using a Tunnel via the UI (see page 808)
- Attach a Cluster Using a Tunnel via the CLI (see page 819)
- API documentation (v1alpha1) (see page 835)
- Proxied Access to Network-Restricted Clusters (see page 846)

5.5.4.5.4 Prerequisites for a Tunneled Attachment

Enterprise Gov Advanced

5.5.4.5.4.1 Before you Begin

- Gain more understanding of this approach by reviewing Attach a Cluster with Networking Restrictions (see page 804).
- Ensure you have reviewed the general Requirements for Attaching an Existing Cluster (see page 785).

5.5.4.5.4.2 Prerequisites

To enable a tunneled attachment, you have the following **additional** prerequisites:

- Ensure that `kubetunnel` is deployed on the Management Cluster.
  Use the following command to check if `kubetunnel` is deployed:

  ```
  kubectl get appdeployments.apps.kommander.d2iq.io -n kommander kubetunnel
  ```

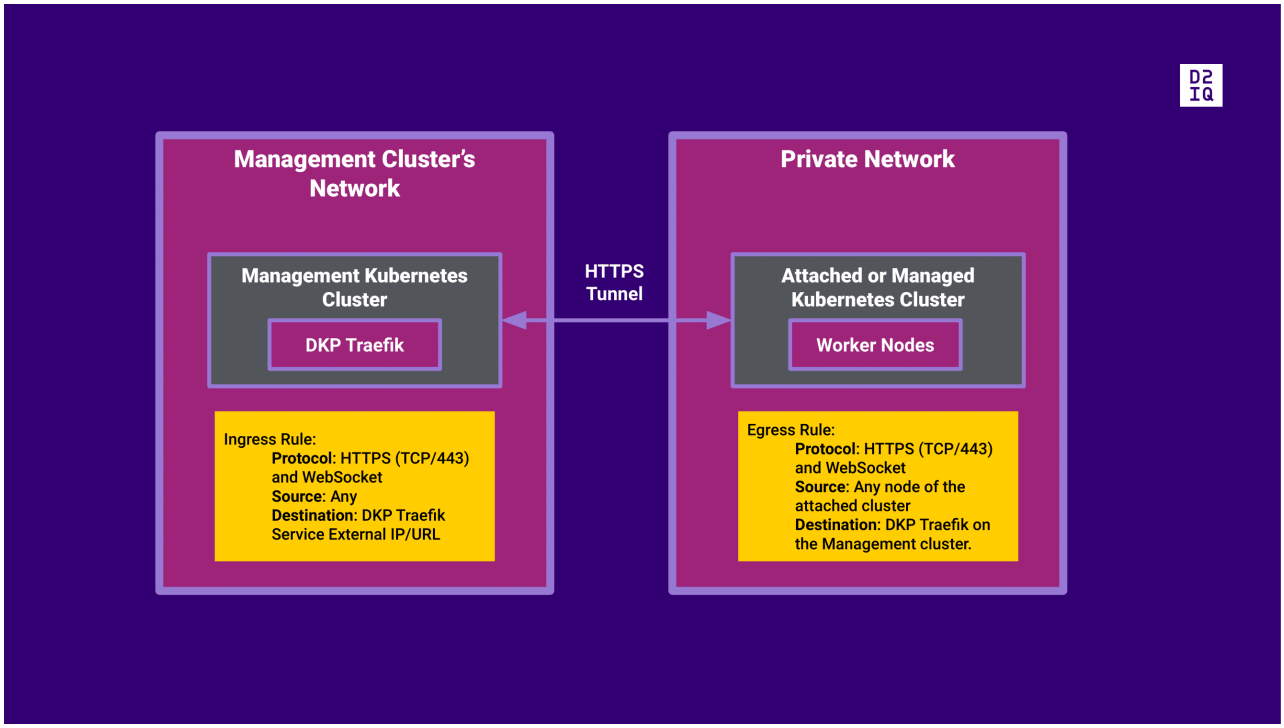
  The output should look similar to this:

  ```
  NAME         APP                 AGE
  kubetunnel   kubetunnel-<version>   5h14m
  ```

> Kubetunnel is deployed by default. If you need to install it manually, see Deploy Platform Applications via CLI (see page 667) and ensure you install it on the Management cluster.

- **Firewall rules**:

	The ingress rule on the Management cluster network must allow:	**The egress rule on the Attached or Managed cluster private network must allow:**
Protocol	HTTPS (TCP/443) and WebSocket	HTTPS (TCP/443) and WebSocket
Source	Any	Any node of the Attached or Managed cluster
Destination	DKP Traefik Service External IP/URL	DKP Traefik Service on the Management cluster



Choose your Next Step:

Attach a Cluster using a Tunnel via the UI (see page 808)

Attach a Cluster Using a Tunnel via the CLI (see page 819)

5.5.4.5.5 Attach a Cluster using a Tunnel via the UI

Use the UI to attach a network-restricted cluster (see page 82)

> Ensure you have reviewed and followed the steps in Prerequisites for a Tunneled Attachment (see page 806) before proceeding.

5.5.4.5.5.1 Attach a Cluster Using a Tunnel

Here is a high-level overview of the steps required to attach a cluster via the UI:

- UI: Attach a Network-Restricted Cluster (see page 809)
- UI: Finish Attaching the Existing Cluster (see page 811)
- Use the Network-restricted Cluster (see page 812)

5.5.4.5.5.2 UI: Attach a Network-Restricted Cluster

Attach a Cluster

To attach a network-restricted cluster (see page 82) to your DKP landscape:

1. From the top menu bar, select your target workspace.
2. On the Dashboard page, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
3. Select **Attach Cluster**.
4. Select the **Cluster has networking restrictions** card to display the configuration page.

Configure the Attachment

Establish the configuration parameters for the attachment:

5. Enter the **Cluster Name** of the cluster you're attaching.
6. Create additional new **Labels** as needed.
7. Select the hostname that is the Ingress for the cluster from the **Load Balancer Hostname** dropdown menu. The hostname must match the Kommander Host cluster to which you are attaching your existing cluster with network restrictions.
8. Specify the **URL Path Prefix** for your Load Balancer Hostname. This URL path will serve as the prefix for the specific tunnel services you want to expose on the Kommander management cluster. If no value is specified, the value defaults to `/dkp/tunnel`.

   Kommander uses Traefik 2 ingress, which requires explicit definition of strip prefix middleware as a Kubernetes API object, opposed to a simple annotation. Kommander provides default middleware that supports creating tunnels only on the `/dkp/tunnel URL` prefix. This is indicated by using the extra annotation, `traefik.ingress.kubernetes.io/router.middlewares: kommander-stripprefixes-kubetunnel@kubernetescrd` as shown in the code sample that follows. If you want to expose a tunnel on a different URL prefix, you must manage your own middleware configuration.
9. **Optional:** Enter a value for the **Hostname** field.
10. Provide a secret for your certificate in the **Root CA Certificate** drop-down menu.
    a. For environments where the Management cluster uses a publicly-signed CA (like ZeroSSL or Let's Encrypt), select **Use Publicly Trusted CA**.
    b. If you manually created a secret in advance, select it from the drop-down menu.
    c. For all other cases, select **Create a new secret**. Then, execute the following command on the Management cluster to obtain the `caBundle` key:

```
kubectl get kommandercluster -n kommander host-cluster -o go-
template='{{ .status.ingress.caBundle }}'
```

Copy and paste the output into the **Root CA Certificate** field.

11. Add any **Extra Annotations** as needed.

Optional: Enable a Proxied Access

Activate a proxied access (see page 846) to enable kubectl access and dashboard observability for the network-restricted cluster from the Management cluster (see page 80). For more information, see Proxied Access to Network-Restricted Clusters (see page 846).

12. Select **Show Advanced**.
13. Add a **Cluster Proxy Domain**.

- If you previously configured a domain wildcard (see page 855) for your cluster, a Cluster Proxy Domain is suggested automatically based on your cluster name. Replace the suggestion if you want to assign a different domain for the proxied cluster.
- If you want to use the `external-dns` service, specify a **Cluster Proxy Domain** that is within the zones specified in the `--domain-filter` argument of the external-dns deployment manifest (see page 1579) stored on the **Management cluster**.
  For example, if the filter is set to `example.com`, a possible domain for the `TUNNEL_PROXY_EXTERNAL_DOMAIN` would be `myclusterproxy.example.com`.

14. Establish a DNS record and certificate configuration for the **Cluster Proxy Domain.** You can choose between the default and a custom option:

	DNS record creation	**Certificate Management**
***Default settings* box checked ✓**	Automatic, handled by `external-dns`	Automatic, handled by `kommander-ca`

***Custom settings* box unchecked ☐**	Manually create a DNS record. The record's A/CNAME value must point to the Management cluster's Traefik IP address, URL or domain. **OR** Enable `external-dns` with an annotation that points to the **Cluster Proxy Domain**.	Select an existing **TLS** certificate. **OR** Select an existing **Issuer** or **ClusterIssuer**.

15. Select the **Save & Generate kubeconfig** button to generate a file required to finish attaching the cluster.

A new window appears with instructions on how to finalize attaching the cluster. See UI: Finish Attaching the Existing Cluster (see page 811) for further instrucions.

Next Step:

UI: Finish Attaching the Existing Cluster (see page 811)

Related Topic:

For information on the TunnelGateway review the API documentation (v1alpha1) (see page 835).

5.5.4.5.5.3 UI: Finish Attaching the Existing Cluster

Enterprise Gov Advanced

How to apply the kubeconfig file to create the network tunnel to attach a network-restricted cluster (see page 82)

After you have configured your cluster's attachment in UI: Attach a Network-Restricted Cluster (see page 809), finalize attaching the cluster. Now you must apply the generated manifest to create the network tunnel and complete the attachment process:

1. Select the **Download Manifest** link to download the file you generated previously (see page 809).
2. Copy the `kubectl apply` command from the UI and paste into your terminal session, DO NOT run it yet.
3. Ensure you substitute the actual name of the file for the variable. Also ensure you use the `--kubeconfig=<managed_cluster_kubeconfig.conf>` flag to run the command on the Attached or Managed cluster. Run the command.
   :note: Running this command starts the attachment process, which may take several minutes to complete. The Cluster details page will be displayed automatically when the cluster attachment process completes.

4. **Optional:** Select the **Verify Connection to Cluster** button to send a request to Kommander to refresh the connection information. You can use this option to check to see if the connection is complete, though the Cluster Details page displays automatically when the connection is complete.

> After the initial connection is made, and your cluster becomes viewable as attached in the DKP UI, the attachment, federated add-ons, and platform services will still need to be completed. This may take several additional minutes. If a cluster has limited resources to deploy all the federated platform services, the installation of the federated resources will fail, and the cluster may become unreachable in the DKP UI. If this happens, check whether there are any pods that are not getting the resources required.

Next Step:

Now you can learn how to Use the Network-restricted Cluster (see page 812) (this approach uses the CLI).

5.5.4.5.5.4 Use the Network-restricted Cluster

Enterprise Gov Advanced

To access services running on the remote, edge or network-restricted cluster (see page 82) from the Management cluster, connect to the tunnel proxy.

You can use these three methods:

1. If the client program supports use of a kubeconfig file, use the network-restricted cluster's kubeconfig.
2. If the client program supports SOCKS5 proxies, use the proxy directly.
3. Otherwise, deploy a proxy server on the Management cluster.

Network-restricted Cluster Service

These sections require a service to run on the Attached or Managed network-restricted cluster.

As an example, start the following service:

```
service_namespace=test
service_name=webserver
service_port=8888
service_endpoint=${service_name}.${service_namespace}.svc.cluster.local:$
{service_port}

cat > nginx.yaml <<EOF
apiVersion: v1
```

```
kind: Namespace
metadata:
  name: ${service_namespace}
---
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: ${service_namespace}
  name: nginx-deployment
  labels:
    app: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx-app
  template:
    metadata:
      labels:
        app: nginx-app
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  namespace: ${service_namespace}
  name: ${service_name}
spec:
  selector:
    app: nginx-app
  type: ClusterIP
  ports:
  - targetPort: 80
    port: ${service_port}
EOF

kubectl apply -f nginx.yaml

kubectl rollout status deploy -n ${service_namespace} nginx-deployment
```

On the Attached or Managed cluster, a client Job can access this service using:

```
cat > curl.yaml <<EOF
apiVersion: batch/v1
kind: Job
metadata:
```

```
  name: curl
spec:
  template:
    spec:
      containers:
      - name: curl
        image: curlimages/curl:7.76.0
        command: ["curl", "--silent", "--show-error", "http://${service_endpoint}"]
      restartPolicy: Never
  backoffLimit: 4
EOF

kubectl apply -f curl.yaml

kubectl wait --for=condition=complete job curl

podname=$(kubectl get pods --selector=job-name=curl --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')

kubectl logs ${podname}
```

The final command returns the default Nginx web page:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Use of kubeconfig file

This is primarily useful for running `kubectl` commands on the Management cluster to monitor the network-restricted, Managed or Attached cluster.

On the Management cluster, a `kubeconfig` file for the Attached or Managed cluster configured to use the tunnel proxy is available as a Secret. The Secret's name can be identified using:

```
kubeconfig_secret=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath='{.status.kubeconfigRef.name}')
```

After setting `service_namespace` and `service_name` to the service resource, run this command on the Management cluster:

```
cat > get-service.yaml <<EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: get-service
spec:
  template:
    spec:
      containers:
      - name: kubectl
        image: bitnami/kubectl:1.19
        command: ["kubectl", "get", "service", "-n", "${service_namespace}", "$
{service_name}"]
        env:
        - name: KUBECONFIG
          value: /tmp/kubeconfig/kubeconfig
        volumeMounts:
        - name: kubeconfig
          mountPath: /tmp/kubeconfig
      volumes:
      - name: kubeconfig
        secret:
          secretName: "${kubeconfig_secret}"
      restartPolicy: Never
  backoffLimit: 4
EOF

kubectl apply -n ${namespace} -f get-service.yaml

kubectl wait --for=condition=complete --timeout=5m job -n ${namespace} get-service

podname=$(kubectl get pods -n ${namespace} --selector=job-name=get-service --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')

kubectl logs -n ${namespace} ${podname}
```

Direct use of SOCKS5 proxy

To use the SOCKS5 proxy directly, obtain the SOCKS5 proxy endpoint using:

```
proxy_service=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath='{.status.tunnelServer.serviceRef.name}')

socks_proxy=$(kubectl get service -n ${namespace} "${proxy_service}" -o jsonpath='{.s
pec.clusterIP}{":"}{.spec.ports[?(@.name=="proxy")].port}')
```

Provide the value of `${socks_proxy}` as the SOCKS5 proxy to your client.

For example, since `curl` supports SOCKS5 proxies, the Attached or Managed service started above can be accessed from the Management cluster by adding the SOCKS5 proxy to the `curl` command. After setting `service_endpoint` to the service endpoint, on the Management cluster run:

```
cat > curl.yaml <<EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: curl
spec:
  template:
    spec:
      containers:
      - name: curl
        image: curlimages/curl:7.76.0
        command: ["curl", "--silent", "--show-error", "--socks5-hostname", "$
{socks_proxy}", "http://${service_endpoint}"]
      restartPolicy: Never
  backoffLimit: 4
EOF

kubectl apply -f curl.yaml

kubectl wait --for=condition=complete --timeout=5m job curl

podname=$(kubectl get pods --selector=job-name=curl --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')

kubectl logs ${podname}
```

The final command returns the same output as for the job on the Attached or Managed cluster, demonstrating that the job on the Management cluster accessed the service running on the Attached or Managed cluster.

Use of deployed proxy on Management cluster

To deploy a proxy on the Management cluster, obtain the SOCKS5 proxy endpoint using:

```
proxy_service=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath='{.status.tunnelServer.serviceRef.name}')

socks_proxy=$(kubectl get service -n ${namespace} "${proxy_service}" -o jsonpath='{.s
pec.clusterIP}{":"}{.spec.ports[?(@.name=="proxy")].port}')
```

Provide the value of `${socks_proxy}` as the SOCKS5 proxy to a proxy deployed on the Management cluster. After setting `service_endpoint` to the service endpoint, on the Management cluster run:

```
cat > nginx-proxy.yaml <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: nginx-proxy-crt
spec:
  secretName: nginx-proxy-crt-secret
  dnsNames:
  - nginx-proxy-service.${namespace}.svc.cluster.local
  issuerRef:
    group: cert-manager.io
    kind: ClusterIssuer
    name: kubernetes-ca
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-proxy
  labels:
    app: nginx-proxy-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-proxy-app
  template:
    metadata:
      labels:
        app: nginx-proxy-app
    spec:
      containers:
      - name: nginx-proxy
        image: mesosphere/ghostunnel:v1.5.3-server-backend-proxy
        args:
        - "server"
        - "--listen=:443"
        - "--target=${service_endpoint}"
        - "--cert=/etc/certs/tls.crt"
```

```
        - "--key=/etc/certs/tls.key"
        - "--cacert=/etc/certs/ca.crt"
        - "--unsafe-target"
        - "--disable-authentication"
        env:
        - name: ALL_PROXY
          value: socks5://${socks_proxy}
        ports:
        - containerPort: 443
        volumeMounts:
        - name: certs
          mountPath: /etc/certs
      volumes:
      - name: certs
        secret:
          secretName: nginx-proxy-crt-secret
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-proxy-service
spec:
  selector:
    app: nginx-proxy-app
  type: ClusterIP
  ports:
  - targetPort: 443
    port: 8765
EOF

kubectl apply -n ${namespace} -f nginx-proxy.yaml

kubectl rollout status deploy -n ${namespace} nginx-proxy

proxy_port=$(kubectl get service -n ${namespace} nginx-proxy-service -o
jsonpath='{.spec.ports[0].port}')
```

Any client running on the Management cluster can now access the service running on the Attached or Managed cluster using the proxy service endpoint. Note in the following that the `curl` job runs in the same namespace as the proxy, to provide access to the CA certificate secret.

```
cat > curl.yaml <<EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: curl
spec:
  template:
    spec:
      containers:
      - name: curl
```

```
        image: curlimages/curl:7.76.0
        command:
        - curl
        - --silent
        - --show-error
        - --cacert
        - /etc/certs/ca.crt
        - https://nginx-proxy-service.${namespace}.svc.cluster.local:${proxy_port}
        volumeMounts:
        - name: certs
          mountPath: /etc/certs
      volumes:
      - name: certs
        secret:
          secretName: nginx-proxy-crt-secret
      restartPolicy: Never
  backoffLimit: 4
EOF

kubectl apply -n ${namespace} -f curl.yaml

kubectl wait --for=condition=complete --timeout=5m job -n ${namespace} curl

podname=$(kubectl get pods -n ${namespace} --selector=job-name=curl --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')

kubectl logs -n ${namespace} ${podname}
```

The final command returns the same output as the job on the Attached or Managed cluster, demonstrating that the job on the Management cluster accessed the service running on the network-restricted cluster.

Related Topic:

For information on the TunnelGateway review the API documentation (v1alpha1) (see page 835).

5.5.4.5.6 Attach a Cluster Using a Tunnel via the CLI

Use the CLI to attach a network-restricted cluster (see page 82) with a Tunnel

Enterprise Gov Advanced

Ensure you have reviewed and followed the steps in Prerequisites for a Tunneled Attachment (see page 806) before proceeding.

5.5.4.5.6.1 Attach a Cluster Using a Tunnel

Here is a high-level overview of the steps required to attach a cluster via the CLI:

- CLI: Prepare the Management Cluster (see page 820)
- CLI: Create and Configure the Tunnel (see page 822)
- CLI: Use the Network-Restricted Cluster (see page 827)

5.5.4.5.6.2 CLI: Prepare the Management Cluster

Enterprise Gov Advanced

Identify the Management Cluster Endpoint

Execute the following command on the Management cluster to obtain the hostname and CA certificate:

```
hostname=$(kubectl get service -n kommander kommander-traefik -o go-template='{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}')
b64ca_cert=$(kubectl get secret -n cert-manager kommander-ca -o=go-
template='{{index .data "tls.crt"}}')
```

Specify a Workspace Namespace

Obtain the desired workspace namespace on the Management cluster for the tunnel gateway:

```
namespace=$(kubectl get workspace default-workspace -o jsonpath="{.status.namespaceRe
f.name}")
```

Alternatively, you can create a new workspace instead of using an existing workspace:
Run the following command, and replace the `<workspace_name>` with the new workspace name:

```
workspace=<workspace_name>
```

Finish creating the workspace:

```
namespace=${workspace}

cat > workspace.yaml <<EOF
apiVersion: workspaces.kommander.mesosphere.io/v1alpha1
kind: Workspace
metadata:
```

```
  annotations:
    kommander.mesosphere.io/display-name: ${workspace}
  name: ${workspace}
spec:
  namespaceName: ${namespace}
EOF

kubectl apply -f workspace.yaml
```

You can verify the workspace exists using:

```
kubectl get workspace ${workspace}
```

Create a Tunnel Gateway

Create a tunnel gateway (see page 0) on the Management cluster to listen for tunnel agents on remote clusters:

> Kommander uses Traefik 2 ingress, which requires explicit definition of strip prefix middleware as a Kubernetes API object, opposed to a simple annotation. Kommander provides default middleware that supports creating tunnels only on the `/dkp/tunnel URL` prefix. This is indicated by using the extra annotation, `traefik.ingress.kubernetes.io/router.middlewares: kommander-stripprefixes-kubetunnel@kubernetescrd` as shown in the code sample that follows. If you want to expose a tunnel on a different URL prefix, you must manage your own middleware configuration.

Establish variables for the certificate secret and gateway. Replace the `<gateway_name>` placeholder with the name of the gateway:

```
cacert_secret=kubetunnel-ca
gateway=<gateway_name>
```

Create the `Secret` and `TunnelGateway` objects:

```
cat > gateway.yaml <<EOF
apiVersion: v1
kind: Secret
metadata:
  namespace: ${namespace}
  name: ${cacert_secret}
data:
  ca.crt:
```

```
    ${b64ca_cert}
---
apiVersion: kubetunnel.d2iq.io/v1alpha1
kind: TunnelGateway
metadata:
  namespace: ${namespace}
  name: ${gateway}
spec:
  ingress:
    caSecretRef:
      namespace: ${namespace}
      name: ${cacert_secret}
    loadBalancer:
      hostname: ${hostname}
    urlPathPrefix: /dkp/tunnel
    extraAnnotations:
      kubernetes.io/ingress.class: kommander-traefik
      traefik.ingress.kubernetes.io/router.tls: "true"
      traefik.ingress.kubernetes.io/router.middlewares: kommander-stripprefixes-
kubetunnel@kubernetescrd
EOF

kubectl apply -f gateway.yaml
```

You can verify the gateway exists using the command:

```
kubectl get tunnelgateway -n ${namespace} ${gateway}
```

Next Step:

CLI: Create and Configure the Tunnel (see page 822)

5.5.4.5.6.3 CLI: Create and Configure the Tunnel

Enterprise Gov Advanced

Connect a remote, edge or network-restricted cluster

Create a Tunnel Connector

Create a tunnel connector (see page 842) on the **Management cluster** for the remote cluster.

1. Establish a variable for the connector. Provide the name of the connector, by replacing the `<connector_name>` placeholder:

   ```
   connector=<connector_name>
   ```

2. Create the `TunnelConnector` object:

```
cat > connector.yaml <<EOF
apiVersion: kubetunnel.d2iq.io/v1alpha1
kind: TunnelConnector
metadata:
  namespace: ${namespace}
  name: ${connector}
spec:
  gatewayRef:
    name: ${gateway}
EOF

kubectl apply -f connector.yaml
```

After you create the `TunnelConnector` object, DKP creates a `manifest.yaml`. This `manifest.yaml` contains the configuration information for the components required by the tunnel for a specific cluster.

3. Verify the connector exists:

```
kubectl get tunnelconnector -n ${namespace} ${connector}
```

4. Wait for the tunnel connector to reach the `Listening` state and export the agent manifest:

```
while [ "$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath="{.status.state}")" != "Listening" ]
do
  sleep 5
done

manifest=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath="{.status.tunnelAgent.manifestsRef.name}")
while [ -z ${manifest} ]
do
  sleep 5
  manifest=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath="{.status.tunnelAgent.manifestsRef.name}")
done
```

The `manifest.yaml` is applied successfully after the command completes.

5. Fetch the `manifest.yaml` to use it in the following section:

```
kubectl get secret -n ${namespace} ${manifest} -o
jsonpath='{.data.manifests\.yaml}' | base64 -d > manifest.yaml
```

When attaching several clusters, ensure that you fetch the `manifest.yaml` of the cluster you are attempting to attach. Using the wrong combination of `manifest.yaml` and cluster will cause the attachment to fail.

Set up the Network-restricted Cluster

In the following commands, the `--kubeconfig` flag ensures that you set the context to the Attached or Managed cluster. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 106).

1. Apply the `manifest.yaml` file to the **Attached or Managed** cluster and deploy the tunnel agent:

   ```
   kubectl apply --kubeconfig=<managed_cluster_kubeconfig.conf> -f manifest.yaml
   ```

2. Check the status of the created pods using:

   ```
   kubectl get pods --kubeconfig=<managed_cluster_kubeconfig.conf> -n kubetunnel
   ```

   After a short time, expect to see a `post-kubeconfig` pod that reaches `Completed` state and a `tunnel-agent` pod that stays in `Running` state.

   ```
   NAME                            READY   STATUS      RESTARTS   AGE
   post-kubeconfig-j2ghk           0/1     Completed   0          14m
   tunnel-agent-f8d9f4cb4-thx8h    1/1     Running     0          14m
   ```

Add the Network-restricted Cluster into Kommander

When you create a cluster using the DKP CLI, it does not attach automatically.

1. On the Management cluster, wait for the tunnel to be connected by the tunnel agent:

   ```
   while [ "$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
   jsonpath="{.status.state}")" != "Connected" ]
   do
     sleep 5
   ```

```
done
```

2. Establish variables for the managed cluster. Replace the `<private_cluster>` placeholder with the name of the managed cluster:

```
managed=<private-cluster>
display_name=${managed}
```

3. Update the `KommanderCluster` object:

```
cat > kommander.yaml <<EOF
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  namespace: ${namespace}
  name: ${managed}
  annotations:
    kommander.mesosphere.io/display-name: ${display_name}
spec:
  clusterTunnelConnectorRef:
    name: ${connector}
EOF

kubectl apply -f kommander.yaml
```

4. Wait for the Attached or Managed cluster to join:

```
while [ "$(kubectl get kommandercluster -n ${namespace} ${managed} -o
jsonpath='{.status.phase}')" != "Joined" ]
do
  sleep 5
done

kubefed=$(kubectl get kommandercluster -n ${namespace} ${managed} -o
jsonpath="{.status.kubefedclusterRef.name}")
while [ -z "${kubefed}" ]
do
  sleep 5
  kubefed=$(kubectl get kommandercluster -n ${namespace} ${managed} -o
jsonpath="{.status.kubefedclusterRef.name}")
done

kubectl wait --for=condition=ready --timeout=60s kubefedcluster -n kube-
federation-system ${kubefed}
```

After the command completes, your cluster becomes visible in the DKP UI and you can start using it. Its metrics will be accessible through different dashboards such as Grafana, Karma, etc.

Create a Network Policy for the Tunnel Server

This step is optional but improves **security** by restricting which remote hosts can connect to the tunnel.

1. Apply a network policy that restricts tunnel access to specific namespaces and IP blocks.
   The following *example* permits connections from
   - Pods running in the `kommander` and `kube-federation-system` namespace.
   - Remote clusters with IP addresses in the ranges 192.0.2.0 to 192.0.2.255 and 203.0.113.0 to 203.0.113.255.
   - Pods running in namespaces with a label `kubetunnel.d2iq.io/networkpolicy` that match the tunnel name and namespace.

```
cat > net.yaml <<EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  namespace: ${namespace}
  name: ${connector}-deny
  labels:
    kubetunnel.d2iq.io/tunnel-connector: ${connector}
    kubetunnel.d2iq.io/networkpolicy-type: "tunnel-server"
spec:
  podSelector:
    matchLabels:
      kubetunnel.d2iq.io/tunnel-connector: ${connector}
  policyTypes:
  - Ingress
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  namespace: ${namespace}
  name: ${connector}-allow
  labels:
    kubetunnel.d2iq.io/tunnel-connector: ${connector}
    kubetunnel.d2iq.io/networkpolicy-type: "tunnel-server"
spec:
  podSelector:
    matchLabels:
      kubetunnel.d2iq.io/tunnel-connector: ${connector}
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: "kube-federation-system"
    - namespaceSelector:
```

```
        matchLabels:
          kubernetes.io/metadata.name: "kommander"
    - namespaceSelector:
        matchLabels:
          kubetunnel.d2iq.io/networkpolicy: ${connector}-${namespace}
    - ipBlock:
        cidr: 192.0.2.0/24
    - ipBlock:
        cidr: 203.0.113.0/24
EOF

kubectl apply -f net.yaml
```

2. To enable applications running in another namespace to access the attached cluster, add the label `kubetunnel.d2iq.io/networkpolicy=${connector}-${namespace}` to the target namespace:

   ```
   kubectl label ns ${namespace} kubetunnel.d2iq.io/networkpolicy=${connector}-$
   {namespace}
   ```

   All pods in the target namespace can now reach the attached cluster services.

Next Steps:

- **Optional:** If you want to access the network-restricted attached cluster from the Management cluster, enable proxied access (see page 848).
- **Alternatively**, start using your remote cluster (see page 812).

5.5.4.5.6.4 CLI: Use the Network-Restricted Cluster

Enterprise Gov Advanced

To access services running on the remote, edge or network-restricted cluster (see page 82) from the Management cluster, connect to the tunnel proxy.

You can use these three methods:

1. If the client program supports use of a kubeconfig file, use the network-restricted cluster's kubeconfig.
2. If the client program supports SOCKS5 proxies, use the proxy directly.
3. Otherwise, deploy a proxy server on the Management cluster.

Network-restricted Cluster Service

These sections require a service to run on the Attached or Managed network-restricted cluster.

As an example, start the following service:

```
service_namespace=test
service_name=webserver
service_port=8888
service_endpoint=${service_name}.${service_namespace}.svc.cluster.local:$
{service_port}

cat > nginx.yaml <<EOF
apiVersion: v1
kind: Namespace
metadata:
  name: ${service_namespace}
---
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: ${service_namespace}
  name: nginx-deployment
  labels:
    app: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx-app
  template:
    metadata:
      labels:
        app: nginx-app
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  namespace: ${service_namespace}
  name: ${service_name}
spec:
  selector:
    app: nginx-app
  type: ClusterIP
  ports:
  - targetPort: 80
    port: ${service_port}
EOF

kubectl apply -f nginx.yaml
```

```
kubectl rollout status deploy -n ${service_namespace} nginx-deployment
```

On the Attached or Managed cluster, a client Job can access this service using:

```
cat > curl.yaml <<EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: curl
spec:
  template:
    spec:
      containers:
      - name: curl
        image: curlimages/curl:7.76.0
        command: ["curl", "--silent", "--show-error", "http://${service_endpoint}"]
      restartPolicy: Never
  backoffLimit: 4
EOF

kubectl apply -f curl.yaml

kubectl wait --for=condition=complete job curl

podname=$(kubectl get pods --selector=job-name=curl --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')

kubectl logs ${podname}
```

The final command returns the default Nginx web page:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
```

```
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Use of kubeconfig file

This is primarily useful for running `kubectl` commands on the Management cluster to monitor the network-restricted, Managed or Attached cluster.

On the Management cluster, a `kubeconfig` file for the Attached or Managed cluster configured to use the tunnel proxy is available as a Secret. The Secret's name can be identified using:

```
kubeconfig_secret=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath='{.status.kubeconfigRef.name}')
```

After setting `service_namespace` and `service_name` to the service resource, run this command on the Management cluster:

```
cat > get-service.yaml <<EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: get-service
spec:
  template:
    spec:
      containers:
      - name: kubectl
        image: bitnami/kubectl:1.19
        command: ["kubectl", "get", "service", "-n", "${service_namespace}", "$
{service_name}"]
        env:
        - name: KUBECONFIG
          value: /tmp/kubeconfig/kubeconfig
        volumeMounts:
        - name: kubeconfig
          mountPath: /tmp/kubeconfig
      volumes:
      - name: kubeconfig
        secret:
          secretName: "${kubeconfig_secret}"
      restartPolicy: Never
  backoffLimit: 4
EOF
```

```
kubectl apply -n ${namespace} -f get-service.yaml

kubectl wait --for=condition=complete --timeout=5m job -n ${namespace} get-service

podname=$(kubectl get pods -n ${namespace} --selector=job-name=get-service --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')

kubectl logs -n ${namespace} ${podname}
```

Direct use of SOCKS5 proxy

To use the SOCKS5 proxy directly, obtain the SOCKS5 proxy endpoint using:

```
proxy_service=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath='{.status.tunnelServer.serviceRef.name}')

socks_proxy=$(kubectl get service -n ${namespace} "${proxy_service}" -o jsonpath='{.s
pec.clusterIP}{":"}{.spec.ports[?(@.name=="proxy")].port}')
```

Provide the value of `${socks_proxy}` as the SOCKS5 proxy to your client.

For example, since `curl` supports SOCKS5 proxies, the Attached or Managed service started above can be accessed from the Management cluster by adding the SOCKS5 proxy to the `curl` command. After setting `service_endpoint` to the service endpoint, on the Management cluster run:

```
cat > curl.yaml <<EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: curl
spec:
  template:
    spec:
      containers:
      - name: curl
        image: curlimages/curl:7.76.0
        command: ["curl", "--silent", "--show-error", "--socks5-hostname", "$
{socks_proxy}", "http://${service_endpoint}"]
      restartPolicy: Never
  backoffLimit: 4
EOF

kubectl apply -f curl.yaml

kubectl wait --for=condition=complete --timeout=5m job curl

podname=$(kubectl get pods --selector=job-name=curl --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')
```

```
kubectl logs ${podname}
```

The final command returns the same output as for the job on the Attached or Managed cluster, demonstrating that the job on the Management cluster accessed the service running on the Attached or Managed cluster.

Use of deployed proxy on Management cluster

To deploy a proxy on the Management cluster, obtain the SOCKS5 proxy endpoint using:

```
proxy_service=$(kubectl get tunnelconnector -n ${namespace} ${connector} -o
jsonpath='{.status.tunnelServer.serviceRef.name}')

socks_proxy=$(kubectl get service -n ${namespace} "${proxy_service}" -o jsonpath='{.s
pec.clusterIP}{":"}{.spec.ports[?(@.name=="proxy")].port}')
```

Provide the value of `${socks_proxy}` as the SOCKS5 proxy to a proxy deployed on the Management cluster. After setting `service_endpoint` to the service endpoint, on the Management cluster run:

```
cat > nginx-proxy.yaml <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: nginx-proxy-crt
spec:
  secretName: nginx-proxy-crt-secret
  dnsNames:
  - nginx-proxy-service.${namespace}.svc.cluster.local
  issuerRef:
    group: cert-manager.io
    kind: ClusterIssuer
    name: kubernetes-ca
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-proxy
  labels:
    app: nginx-proxy-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-proxy-app
  template:
    metadata:
      labels:
```

```
        app: nginx-proxy-app
    spec:
      containers:
      - name: nginx-proxy
        image: mesosphere/ghostunnel:v1.5.3-server-backend-proxy
        args:
        - "server"
        - "--listen=:443"
        - "--target=${service_endpoint}"
        - "--cert=/etc/certs/tls.crt"
        - "--key=/etc/certs/tls.key"
        - "--cacert=/etc/certs/ca.crt"
        - "--unsafe-target"
        - "--disable-authentication"
        env:
        - name: ALL_PROXY
          value: socks5://${socks_proxy}
        ports:
        - containerPort: 443
        volumeMounts:
        - name: certs
          mountPath: /etc/certs
      volumes:
      - name: certs
        secret:
          secretName: nginx-proxy-crt-secret
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-proxy-service
spec:
  selector:
    app: nginx-proxy-app
  type: ClusterIP
  ports:
  - targetPort: 443
    port: 8765
EOF

kubectl apply -n ${namespace} -f nginx-proxy.yaml

kubectl rollout status deploy -n ${namespace} nginx-proxy

proxy_port=$(kubectl get service -n ${namespace} nginx-proxy-service -o
jsonpath='{.spec.ports[0].port}')
```

Any client running on the Management cluster can now access the service running on the Attached or Managed cluster using the proxy service endpoint. Note in the following that the `curl` job runs in the same namespace as the proxy, to provide access to the CA certificate secret.

```
cat > curl.yaml <<EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: curl
spec:
  template:
    spec:
      containers:
      - name: curl
        image: curlimages/curl:7.76.0
        command:
        - curl
        - --silent
        - --show-error
        - --cacert
        - /etc/certs/ca.crt
        - https://nginx-proxy-service.${namespace}.svc.cluster.local:${proxy_port}
        volumeMounts:
        - name: certs
          mountPath: /etc/certs
      volumes:
      - name: certs
        secret:
          secretName: nginx-proxy-crt-secret
      restartPolicy: Never
  backoffLimit: 4
EOF

kubectl apply -n ${namespace} -f curl.yaml

kubectl wait --for=condition=complete --timeout=5m job -n ${namespace} curl

podname=$(kubectl get pods -n ${namespace} --selector=job-name=curl --field-
selector=status.phase=Succeeded -o jsonpath='{.items[0].metadata.name}')

kubectl logs -n ${namespace} ${podname}
```

The final command returns the same output as the job on the Attached or Managed cluster, demonstrating that the job on the Management cluster accessed the service running on the network-restricted cluster.

Next Step:

Optional: If you want to manage the attached cluster from the Management cluster, enable proxied access (see page 848).

Related Topic:

For information on the TunnelGateway review the API documentation (v1alpha1) (see page 835).

5.5.4.5.7 API documentation (v1alpha1)

API Documentation (v1alpha1)

> *This document is automatically generated from the API definition in the code.*

Page Contents

5.5.4.5.7.1 CertificateSpec

CertificateSpec holds settings for the TunnelProxy's ingress exposed Certificate.

Field	Description	Scheme	Required
issuerRef	IssuerRef is a reference to an `Issuer` or `ClusterIssuer` on the target cluster to be used to create a `Certificate` for the cluster's Ingress. If the type is an Issuer, it must be located in the namespace of the TunnelProxy.	IssuerReference (see page 836)	false
certificateSecretRef	CertificateSecretRef is a reference to a secret of type TLS that holds a TLS certificate, private key and CA certificate. The certificate must be valid for the Ingress hostname. The secret must be located in the same namespace of the TunnelProxy.	v1.LocalObjectReference	false

5.5.4.5.7.2 IngressSpec

IngressSpec holds settings for the TunnelProxy managed Ingress.

Field	Description	Scheme	Required
annotations	Annotations allows to set-up important metadata to configure DNS records and certificate generation	map[string]string	false

Field	Description	Scheme	Required
certificate	Certificate holds settings for the cluster's TunnelProxy exposed Certificate.	CertificateSpec (see page 835)	false

5.5.4.5.7.3 IssuerReference

IssuerReference is a reference to an issuer with a given name, kind and group.

Field	Description	Scheme	Required
name	Name of the issuer being referred to.	string	true
kind	Kind of the issuer being referred to.	string	false
group	Group of the issuer being referred to.	string	false

5.5.4.5.7.4 TunnelProxy

Describes the local endpoint for the tunnel. A remote cluster will connect to this endpoint to create a tunnel.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[586]	false
spec		TunnelProxySpec (see page 837)	false
status		TunnelProxyStatus (see page 838)	false

586 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.26/#objectmeta-v1-meta

5.5.4.5.7.5 TunnelProxyList

Contains a list of `TunnelProxy`.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[587]	false
items		[]TunnelProxy (see page 836)	true

5.5.4.5.7.6 TunnelProxySpec

TunnelProxy describes the exposition of an remote KommanderCluster through the Management cluster.

Field	Description	Scheme	Required
clusterProxyDomain	ClusterProxyDomain sets the desired domain to expose the remote cluster. It expects a domain name without scheme and without port. Ex. "traefik.kommander.cluster", "traefik.kommander.cluster" "10.0.0.1"	string	true
caBundle	CABundle is a PEM encoded CA bundle which should be used to verify the TLS connection for the Ingress-served end-entity certificate.	[]byte	false
remoteClusterEndpoint	RemoteClusterEndpoint points to the remote cluster Endpoint (i.e. Traefik instance). It expects a URL without scheme and (optionally) a port. Ex. "traefik.kommander.cluster", "traefik.kommander.cluster:8080" "10.0.0.1"	string	false
ingress	Ingress holds settings for the TunnelProxy managed Ingress for the proxied cluster.	IngressSpec (see page 835)	false

587 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.26/#listmeta-v1-meta

Field	Description	Scheme	Required
tunnelConnectorRef	TunnelConnectorRef points to the TunnelConnector object to be used by the ReverseProxy	v1.LocalObjectReference	true

5.5.4.5.7.7 TunnelProxyStatus

Field	Description	Scheme	Required
conditions	Conditions contains the status conditions of the object.	[]metav1.Condition	false
clusterProxyDomain	ClusterProxyDomain returns the actual domain to reach the remote cluster.	string	true
clientAuthSecretName	ClientAuthSecretName returns the name of the secret containing the mTLS client-auth Certificate Authority.	string	true
reverseProxyReleaseName	ReverseProxyReleaseName returns the name of the Helm release for the ReverseProxy.	string	true

5.5.4.5.7.8 TunnelGateway

Provides an endpoint for remote clusters to connect to the management cluster.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[588]	false
spec		TunnelGatewaySpec (see page 840)	false

588 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.26/#objectmeta-v1-meta

5.5.4.5.7.9 TunnelGatewayIngressSpec

Field	Description	Scheme	Required
loadBalancer	Ingress point for the load-balancer. Traffic intended for the service should be sent to these ingress points. If not specified, the controller will derive from the Ingress record status field.	corev1.LoadBalancerIngress	false
host	Restrict access to requests addressed to a specific host or domain using the `IngressRule` format. Defaults to allow all hosts.	string	false
urlPathPrefix	URL path prefix to prepend to all endpoints. For example, if this field is set to `/ops/portal/kt`, the ingresses created will have URL paths like `/ops/portal/kt/default/cluster1/tunnel-server` and `/ops/portal/kt/default/cluster1/kubeconfig`. Defaults to root path (`/`).	string	false

Field	Description	Scheme	Required
caSecretRef	A secret reference to the root CA required to verify the ingress endpoints. The secret should have type `Opaque` and contain the key `ca.crt`. If not specified, remote hosts will use their system root CA's to verify the endpoints.	corev1.ObjectReference	false
extraAnnotations	Extra annotations to set on the Ingress object.	map[string]string	false

5.5.4.5.7.10 TunnelGatewayList

Contains a list of `TunnelGateway`.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[589]	false
items		[]TunnelGateway	true

5.5.4.5.7.11 TunnelGatewaySpec

If no ingress is set, the services will only be accessible on `localhost`.

589 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.26/#listmeta-v1-meta

Field	Description	Scheme	Required
ingress	Expose services using an Ingress as specified in the `TunnelGatewayIngressSpec`.	TunnelGatewayIngressSpec (see page 839)	false

5.5.4.5.7.12 KubeconfigWebhookStatus

Status of the kubeconfig webhook.

Field	Description	Scheme	Required
deploymentRef	A reference to the deployment for the kubeconfig webhook.	corev1.LocalObjectReference	false
serviceRef	A reference to the service for the kubeconfig webhook.	corev1.LocalObjectReference	false
ingressRef	A reference to the ingress for the kubeconfig webhook.	corev1.LocalObjectReference	false

5.5.4.5.7.13 TunnelAgentStatus

Status of the tunnel agent.

Field	Description	Scheme	Required
manifestsRef	A reference to a secret holding YAML manifests for launching the tunnel agent on the target cluster. The secret is a generic typed secret with filenames as the keys. There might be multiple files in the secret.	corev1.LocalObjectReference	false

5.5.4.5.7.14 TunnelConnector

Describes the local endpoint for the tunnel. A remote cluster will connect to this endpoint to create a tunnel.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[590]	false
spec		TunnelConnectorSpec (see page 843)	false
status		TunnelConnectorStatus (see page 844)	false

5.5.4.5.7.15 TunnelConnectorList

Contains a list of `TunnelConnector`.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[591]	false

590 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.26/#listmeta-v1-meta
591 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.26/#listmeta-v1-meta

Field	Description	Scheme	Required
items		[]TunnelConnector (see page 842)	true

5.5.4.5.7.16 TunnelConnectorSpec

Field	Description	Scheme	Required
gatewayRef	A reference to the `TunnelGateway` object which describes how tunnel services will be exposed outside the current cluster.	corev1.LocalObjectReference	false
proxyPort	The port for the tunnel proxy.	int32	false

5.5.4.5.7.17 TunnelConnectorStatus

Field	Description	Scheme	Required
state	State of the tunnel connector: `Starting` - the initial state; `Listening` - the local tunnel server is waiting for the remote agent to connect; `Pending` - the remote agent has connected but the local proxy is not ready; `Connected` - the tunnel is configured and contact to the remote API server succeeded; `Disconnected` - the tunnel is configured but contact to the remote API server failed; `Failed` - an unexpected error occurred, such as not being able to parse the kubeconfig.	TunnelConnectorState	false
tunnelServer	Status of the tunnel server.	TunnelServerStatus (see page 845)	false
kubeconfigWebhook	Status of the kubeconfig webhook.	KubeconfigWebhookStatus (see page 841)	false
tunnelAgent	Status of the tunnel agent.	TunnelAgentStatus (see page 841)	false
serviceAccountRef	A reference to the service account that will be used for registration (of the tunnel agent) and authentication purpose.	corev1.LocalObjectReference	false

Field	Description	Scheme	Required
roleRef	A reference to the role that will be bound to the service account for authorization purpose.	corev1.LocalObjectReference	false
roleBindingRef	A reference to the rolebinding that will be created to bind the service account and the role.	corev1.LocalObjectReference	false
kubeconfigRef	A reference to the secret holding the KUBECONFIG that the clients can use to talk to the API server of the target cluster when it becomes available.	corev1.LocalObjectReference	false
gatewayObservedGeneration	The generation of the linked TunnelGateway object associated with this object. When the linked TunnelGateway object is updated, a controller will update this status field which will in turn trigger a reconciliation of this object.	int64	false

5.5.4.5.7.18 TunnelServerStatus

Status of the tunnel server.

Field	Description	Scheme	Required
deploymentRef	A reference to the deployment for the tunnel server.	corev1.LocalObjectReference	false

Field	Description	Scheme	Required
serviceRef	A reference to the service for the tunnel server.	corev1.LocalObjectReference	false
ingressRef	A reference to the ingress for the tunnel server.	corev1.LocalObjectReference	false

5.5.4.5.8 Proxied Access to Network-Restricted Clusters

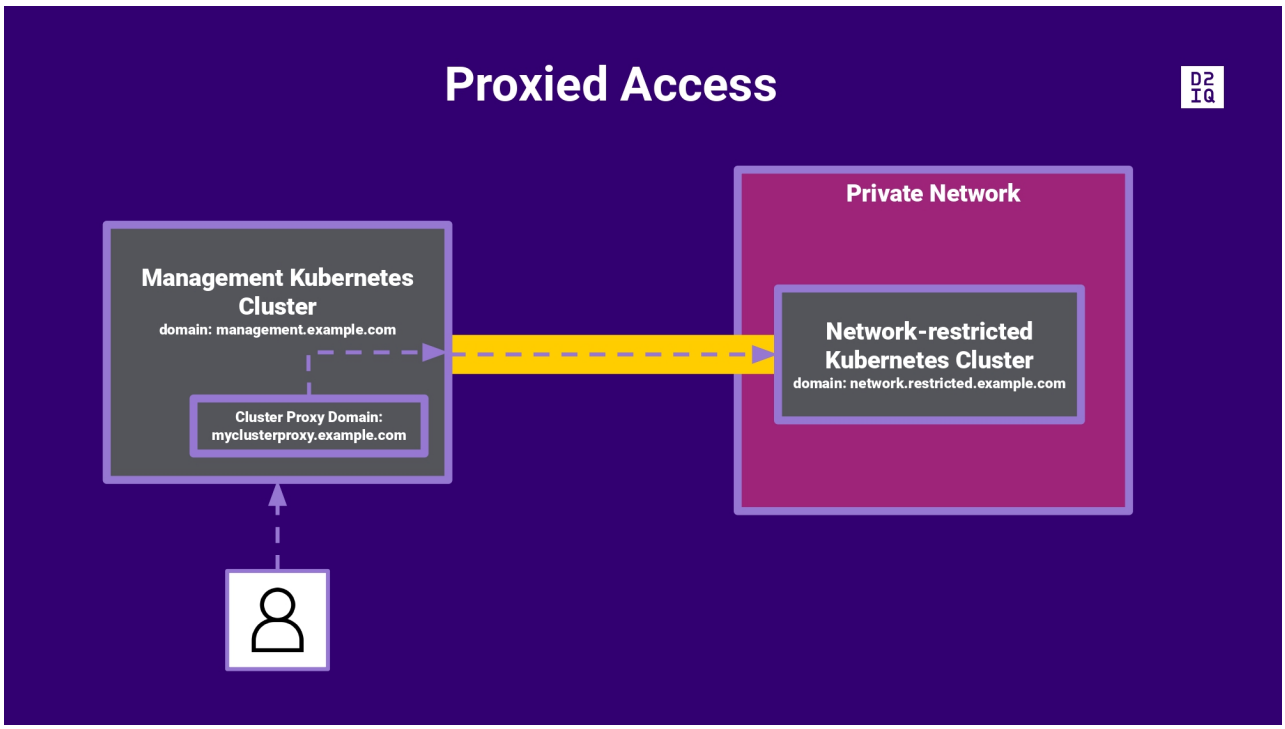
Enterprise Gov Advanced

Enabling a proxied access allows you to access Attached (see page 80) and Managed (see page 80) clusters that are network-restricted (see page 82), in a private network, firewalled, or at edge.

> This section only applies to clusters with networking restrictions that were attached through a secure tunnel (see page 804).

You can attach clusters that are in a private network (see page 804) (clusters that have networking restrictions or are at the edge). D2iQ provides the option of using a secure tunnel or a tunneled attachment to attach a Kubernetes cluster to the Management cluster (see page 80). To **access** these attached clusters through `kubectl` or monitor its resources through the Management cluster, you have to be in the same network, or **enable a proxied access**.

5.5.4.5.8.1 Proxied Access



6 Access to a network-restricted cluster through a proxy in the Management cluster

Enabling the proxied access for a network-restricted cluster (see page 82) makes it possible for DKP to authenticate user requests (regardless of the identity provider (see page 609)) through the Management cluster's authentication proxy. This is helpful, when the cluster you are trying to reach is in a different network. The proxied access allows you to:

- Access and observe the cluster's monitoring and logging services from the Management cluster, for example:
  - Access the cluster's Grafana, Kubernetes and Kubecost dashboards from the Management cluster.
  - Use the CLI to print a cluster's service URL, so you can access the cluster's dashboards.
- Access and perform operations on the network-restricted cluster from the Management cluster, for example:
  - Generate an API token (**Generate Token** option, from the upper right corner of the UI) that allows you to authenticate to the network-restricted cluster.
  - Upon authentication, use `kubectl` to manage the network-restricted cluster.

You can perform the previous actions, without being in the same network as the network-restricted cluster.

5.5.4.5.8.2 Next Step:

Enable Proxied Access Using the CLI (see page 848)

5.5.4.5.8.3 Enable Proxied Access Using the CLI

Enterprise Gov Advanced

Enabling a proxied access allows you to access Attached (see page 80) and Managed (see page 80) clusters that are network-restricted (see page 82), in a private network, firewalled, or at edge.

> This section only applies to clusters with networking restrictions that were attached through a secure tunnel (see page 804).

Continue with the following procedures to enable and configure a proxied access for your cluster:

- 1. Prerequisites: CLI Proxied Access (see page 848)
- 2. Prepare your Environment: CLI Proxied Access (see page 848)
- 3. Create a TunnelProxy Object: CLI Proxied Access (see page 850)
- 4. Enable the TunnelProxy Object in the KommanderCluster: CLI Proxied Access (see page 853)
- 5. Verify the Proxy: CLI Proxied Access (see page 854)

1. Prerequisites: CLI Proxied Access

- You have attached a network-restricted cluster (see page 804).
- The Management (see page 80) and network-restricted cluster are on the same DKP version.
- You have a domain that you can use to put on top of the network-restricted cluster's domain to redirect user requests (Cluster Proxy Domain).
- A DNS record to map your domain to your cluster. There are two supported options for this:
  - **Manual** DNS record creation:
    Create a DNS record manually. The record's A/CNAME value must point to the Management cluster's Traefik IP address, URL or domain. Use one record per proxied cluster.
  - **Automatic** DNS record creation:
    A service that creates and maintains your DNS record automatically. For this method, enable the external-dns service on the Management cluster (see page 1579) before configuring the proxy.

The following pages walk you through enabling the proxied access on the network-restricted cluster.

Next Step:

2. Prepare your Environment: CLI Proxied Access (see page 848)

2. Prepare your Environment: CLI Proxied Access

Establish the following environment variables on the **Management cluster**.

See Provide Context for Commands with a kubeconfig File (see page 106) for more information around switching cluster contexts.

The following commands allow you to run most commands without replacing the information manually.

1. Set the `WORKSPACE_NAMESPACE` environment variable to the name of your network-restricted cluster's workspace namespace:

   ```
   export WORKSPACE_NAMESPACE=<workspace namespace>
   ```

2. Set the variable to the proxy domain through which your cluster should be available:

   ```
   TUNNEL_PROXY_EXTERNAL_DOMAIN=<myclusterproxy.example.com>
   ```

   If you want to use the `external-dns` service, specify a `TUNNEL_PROXY_EXTERNAL_DOMAIN` that is within the zones specified in the `--domain-filter` argument of the external-dns deployment manifest (see page 1579) stored on the **Management cluster**.

   For example, if the filter is set to `example.com`, a possible domain for the `TUNNEL_PROXY_EXTERNAL_DOMAIN` would be `myclusterproxy.example.com`.

3. Establish a variable that points to the name of the network-restricted cluster:
   The name of the network-restricted cluster is established in the KommanderCluster (see page 1567) object.

   ```
   NETWORK_RESTRICTED_CLUSTER=<name_of_restricted_cluster>
   ```

4. Given that each cluster can only have one proxy domain, reuse the name of the network-restricted cluster for the proxy object:

   ```
   TUNNEL_PROXY_NAME=${NETWORK_RESTRICTED_CLUSTER}
   ```

5. Obtain the name of the connector and set it to a variable:

   ```
   TUNNEL_CONNECTOR_NAME=$(kubectl get kommandercluster -n ${WORKSPACE_NAMESPACE}
   ${NETWORK_RESTRICTED_CLUSTER} -o
   template='{{ .spec.clusterTunnelConnectorRef.name }}')
   ```

Next Step:

3. Create a TunnelProxy Object: CLI Proxied Access (see page 850)

3. Create a TunnelProxy Object: CLI Proxied Access

On the **Management cluster**, create a `TunnelProxy` object for your proxied cluster and assign it a unique domain. This domain forwards all user authentication requests through the Management cluster, and is used to generate a URL that exposes the cluster's dashboards (`clusterProxyDomain`).

You require both a **certificate** and a **DNS record** to back the domain. If you choose the default configuration, DKP will handle the certificate creation (self-signed certificate), but you must create a DNS record manually.

Alternatively, you can set up a different Certificate Authority to handle the certificate creation and rotation for your domain. You can also set up the `external-dns` service to automatically create a DNS record.

Here are some examples of possible configuration combinations:

- Ensure you have set the required variables in 2. Prepare your Environment: CLI Proxied Access (see page 848) before you run the following commands.
- Ensure you run the following commands on the Management cluster. See Provide Context for Commands with a kubeconfig File (see page 106) for more information around switching cluster contexts.

Domain with default certificate and automatic DNS record creation (requires External DNS)

In this example, the following configuration applies:

- **Certificate** - The domain uses a self-signed certificate created by DKP.
- **DNS record** - The `external-dns` manages the creation of a DNS record automatically. For it to work, ensure you have enabled External DNS in your Management cluster (see page 1579).

```
cat > tunnelproxy.yaml <<EOF | kubectl apply -f -
apiVersion: kubetunnel.d2iq.io/v1alpha1
kind: TunnelProxy
metadata:
  name: ${TUNNEL_PROXY_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  clusterProxyDomain: ${TUNNEL_PROXY_EXTERNAL_DOMAIN}
  tunnelConnectorRef:
    name: ${TUNNEL_CONNECTOR_NAME}
  ingress:
    annotations:
      external-dns.alpha.kubernetes.io/hostname: ${TUNNEL_PROXY_EXTERNAL_DOMAIN}
```

```
EOF
```

The `spec.ingress.annotations` field contains the annotation required for DNS record management. For more information, see DNS Record Creation with External DNS (see page 1579).

Domain with default certificate and default DNS setup (requires manually-created DNS record)

In this example, the following configuration applies:

- **Certificate** - The domain uses a self-signed certificate created by DKP.
- **DNS record** - For the domain to be recognized by the cluster, ensure you manually create a DNS record. The record's A/CNAME value must point to the Management cluster's Traefik IP address, URL or domain. Create a record per proxied cluster.

```
cat > tunnelproxy.yaml <<EOF | kubectl apply -f -
apiVersion: kubetunnel.d2iq.io/v1alpha1
kind: TunnelProxy
metadata:
  name: ${TUNNEL_PROXY_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  clusterProxyDomain: ${TUNNEL_PROXY_EXTERNAL_DOMAIN}
  tunnelConnectorRef:
    name: ${TUNNEL_CONNECTOR_NAME}
EOF
```

Domain with automatically-generated ACME certificate and automatic DNS record creation (requires External DNS)

In this example, the following configuration applies:

- **Certificate** - The domain uses `cert-manager` to enable an ACME-based Certificate Authority. This CA automatically issues and rotates your certificates. By default, DKP uses Let's Encrypt.
- **DNS record** - The `external-dns` manages the creation of a DNS record automatically. For it to work, ensure you have enabled External DNS in your Management cluster (see page 1579).

1. Set the environment variable for your issuing object:
   This can be a `ClusterIssuer` or `Issuer`. See Advanced Configuration: ClusterIssuer (see page 1574) for more information.

   ```
   ISSUER_KIND=ClusterIssuer
   ```

2. Set the environment variable for your CA:
   Replace `letsEncrypt` if you are using another ACME-based certificate authority.

```
ISSUER_NAME=letsEncrypt
```

3. Create the `TunnelProxy`:

```
cat > tunnelproxy.yaml <<EOF | kubectl apply -f -
apiVersion: kubetunnel.d2iq.io/v1alpha1
kind: TunnelProxy
metadata:
  name: ${TUNNEL_PROXY_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  clusterProxyDomain: ${TUNNEL_PROXY_EXTERNAL_DOMAIN}
  tunnelConnectorRef:
    name: ${TUNNEL_CONNECTOR_NAME}
  ingress:
    annotations:
      external-dns.alpha.kubernetes.io/hostname: $
{TUNNEL_PROXY_EXTERNAL_DOMAIN}
    certificate:
      issuerRef:
        kind: ${ISSUER_KIND}
        name: ${ISSUER_NAME}
EOF
```

For more information, see Configure the Kommander Installation with a Custom Domain and Certificate (see page 1569) and DNS Record Creation with External DNS (see page 1579).

Domain with a custom certificate (requires certificate secret) and automatic DNS record creation (requires External DNS)

In this example, the following configuration applies:

- **Certificate** - The domain uses a custom certificate created manually. Ensure you reference the `<certificate_secret_name>`.
- **DNS record** - The `external-dns` manages the creation of a DNS record automatically. For it to work, ensure you have enabled External DNS in your Management cluster (see page 1579).

1. Set an environment variable for the name of your custom certificate:
   See Manually-generated certificate (see page 1572) for more information.

```
CERTIFICATE_SECRET_NAME=<custom_certificate_secret_name>
```

2. Optional: If you do not have a secret yet and wish to create one pointing at the certificate, execute the following command:

```
kubectl create secret tls ${CERTIFICATE_SECRET_NAME} -n ${WORKSPACE_NAMESPACE}
--key="tls.key" --cert="tls.crt"
```

3. Create the `TunnelProxy`:

```
cat > tunnelproxy.yaml <<EOF | kubectl apply -f -
apiVersion: kubetunnel.d2iq.io/v1alpha1
kind: TunnelProxy
metadata:
  name: ${TUNNEL_PROXY_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  clusterProxyDomain: ${TUNNEL_PROXY_EXTERNAL_DOMAIN}
  tunnelConnectorRef:
    name: ${TUNNEL_CONNECTOR_NAME}
  ingress:
    annotations:
      external-dns.alpha.kubernetes.io/hostname: $
{TUNNEL_PROXY_EXTERNAL_DOMAIN}
    certificate:
      certificateSecretRef:
         name: ${CERTIFICATE_SECRET_NAME}
EOF
```

For more information, see manually-created certificates (see page 892) and DNS Record Creation with External DNS (see page 1579).

Next Step:

4. Enable the TunnelProxy Object in the KommanderCluster: CLI Proxied Access (see page 853)

4. Enable the TunnelProxy Object in the KommanderCluster: CLI Proxied Access

- Ensure you have set the required variables in 2. Prepare your Environment: CLI Proxied Access (see page 848) and created a TunnelProxy object in 3. Create a TunnelProxy Object: CLI Proxied Access (see page 850) before you run the following commands.
- Ensure you run the following command on the Management cluster. See Provide Context for Commands with a kubeconfig File (see page 106) for more information around switching cluster contexts.

To enable the TunnelProxy, reference the object in the KommanderCluster object.

On the **Management cluster**, patch the `KommanderCluster` object with the name of the `TunnelProxy` you created in the previous page:

```
kubectl patch --type merge kommanderclusters -n ${WORKSPACE_NAMESPACE} $
{NETWORK_RESTRICTED_CLUSTER} --patch "{\"spec\": {\"clusterTunnelProxyConnectorRef\":
{ \"name\": \"${TUNNEL_PROXY_NAME}\"}}}"
```

Next Step:

5. Verify the Proxy: CLI Proxied Access (see page 854)

5. Verify the Proxy: CLI Proxied Access

On the **Management cluster**:

> Ensure you run the following command on the Management cluster. See Provide Context for Commands with a kubeconfig File (see page 106) for more information around switching cluster contexts.

1. Verify that the following conditions for the `TunnelProxy` configuration are met:

   ```
   kubectl wait --for=condition=ClientAuthReady=true --timeout=300s -n $
   {WORKSPACE_NAMESPACE} tunnelproxy/${TUNNEL_PROXY_NAME}
   kubectl wait --for=condition=ReverseProxyReady=true --timeout=300s -n $
   {WORKSPACE_NAMESPACE} tunnelproxy/${TUNNEL_PROXY_NAME}
   kubectl wait --for=condition=available -n ${WORKSPACE_NAMESPACE} deploy  -l
   control-plane=${TUNNEL_PROXY_NAME}-kubetunnel-reverse-proxy-rp
   ```

   The output should look like this:

   ```
   tunnelproxy.kubetunnel.d2iq.io/test condition met
   tunnelproxy.kubetunnel.d2iq.io/test condition met
   deployment.apps/${TUNNEL_PROXY_NAME}-kubetunnel-reverse-proxy-rp condition met
   ```

2. Verify that the `TunnelProxy` is correctly assigned and connected to your cluster:

   ```
   curl -Lk -s -o /dev/null -w "%{http_code}" https://$
   {TUNNEL_PROXY_EXTERNAL_DOMAIN}/dkp/grafana
   ```

The output should return a successful HTTP response status:

```
200
```

Now you can access the network-restricted cluster's dashboards and use kubectl to manage its resources from the Management cluster[592].

5.5.4.5.8.4 Enable Proxied Access Using the UI

See UI: Attach a Network-Restricted Cluster (see page 809) for instructions on how to enable a proxied access while you attach a cluster with the UI.

If you attached your Kubernetes cluster to your DKP environment already, it is not possible to enable the proxied access using the UI. Use the CLI as explained in Enable Proxied Access Using the CLI (see page 848).

Configure a Wildcard Domain for your Proxied Cluster

Wildcard domains are helpful in multi-cluster environments. When you set up a wildcard domain, every time you attach an additional network-restricted cluster (see page 82) through a proxy, the DKP UI pre-fills a domain for the cluster automatically.

After you set up a wildcard domain in the `kommander.yaml` as explained in the following section, DKP will suggest domains for attached clusters automatically based on the wildcard domain + the name of the cluster, for example:

Wildcard Domain	Cluster Name	Cluster Domain
*.example.com	development.cluster	development.cluster.example.com
*.example.com	janedoe	janedoe.example.com

You can override the domain by overwriting the suggested domain in the UI.

Set up a Wildcard Domain

1. Open the `kommander.yaml` file:
   a. If you have not installed the Kommander component yet, initialize the configuration file (see page 1558), so you can edit it in the following steps.
   ⚠️ **WARNING**: Initialize this file only ONCE, otherwise you will overwrite previous customizations.
   b. If you have installed the Kommander component already, open the existing `kommander.yaml` with the editor of your choice.

592 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/214630533/DKP+2.6.0+Known+Issues+and+Limitations#GenerateToken

2. Adjust the `apps` section of your `kommander.yaml` file to include these values:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
[...]
  kubetunnel:
    enabled: true
    values: |
      proxiedAccess:
         clusterDomainWildcard: "*.example.com"
```

3. Use the configuration file to install or update the Kommander component:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Whenever you attach a network-restricted cluster, the UI will suggest a new domain based on the wildcard domain and cluster name.

Configure a DNS Record or DNS Service

The clusters will not be available through the established domain until you manually create a DNS record or enable `external-dns` to automatically manage the creation of records.

D2iQ recommends enabling the External DNS (see page 1579) service to manage your records automatically. However, you can choose to create your records manually.

5.5.4.6 Attach a DKP-created Kubernetes Cluster

This section contains the following topics:

- Make a DKP-CLI-created Cluster Managed (see page 856)

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

5.5.4.6.1 Make a DKP-CLI-created Cluster Managed

Enterprise Gov Advanced

5.5.4.6.1.1 Manually Attach a DKP CLI Cluster to the Management Cluster

> ⚠ These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

   ```
   echo ${MANAGED_CLUSTER_NAME}
   ```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
   NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

   ```
   kubectl get workspaces -A
   ```

5. Set the WORKSPACE_NAMESPACE environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace-namespace>
   ```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

   ```
   kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
   template='{{.data.value}}{{ "\n"}}'
   ```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

5.5.4.7 Access a Managed or Attached Cluster

Enterprise Gov Advanced

How to access a Managed or Attached cluster with credentials

5.5.4.7.1 Access your Clusters Using your UI Administrator Credentials

After the cluster is attached, retrieve a custom `kubeconfig` file from the UI.

1. Select the username in the top right corner, and then select **Generate Token**.
2. Select the cluster name, and follow the instructions to assemble a `kubeconfig` for accessing its Kubernetes API.

> If the UI prompts you to log in, use the credentials you normally use to access the UI.

You can also retrieve a custom `kubeconfig` file by visiting the `/token` endpoint on the Kommander cluster domain (example URL: `https://your-server-name.your-region-2.elb.service.com/token/`. Selecting the cluster's name displays the instructions to assemble a `kubeconfig` for accessing its Kubernetes API.

5.5.4.8 Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster

Enterprise Gov Advanced

If you are a DKP Essential customer, you can easily convert your independent clusters into a multi-cluster environment if you upgrade to an Enterprise license.

The pages in this section provides information on how you can turn your Essential Clusters into Managed clusters.

- Prerequisites: General Prerequisites for your Cluster Conversion (see page 860)
- Prerequisites: Prepare your Cluster's Existing Configurations (see page 862)
- Prerequisites: Clone Git Repository from Gitea (see page 863)
- Back up your Cluster's Applications and Persistent Volumes (see page 863)
- UI: Convert an Essential Cluster into a Managed Cluster (see page 874)
- CLI: Convert an Essential Cluster into a Managed Cluster (see page 875)
- Post Conversion Cleanup: Cluster Autoscaler Configuration (see page 877)
- Post Conversion Cleanup: Clusters run on Different Cloud Platforms (see page 880)
- Troubleshooting (see page 881)

For all the instructions regarding converting an existing Essential cluster into an Enterprise cluster, refer to that entire section of documentation (see page 859).

5.5.5 Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster

Enterprise Gov Advanced

If you are a DKP Essential customer, you can easily convert your independent clusters into a multi-cluster environment if you upgrade to an Enterprise license.

The pages in this section provides information on how you can turn your Essential Clusters into Managed clusters.

5.5.5.1 Prerequisites: General Prerequisites for your Cluster Conversion

Information that you need to know prior to converting your DKP Essential Clusters into DKP Enterprise Managed Clusters.

Enterprise Gov Advanced

Starting with DKP 2.5, you have the option to convert your DKP Essential Clusters into DKP Managed Enterprise Clusters.

5.5.5.1.1 Requirements

To convert your DKP Essential clusters into DKP Managed Enterprise clusters, ensure the following is valid:

- A DKP Management cluster with a valid DKP Enterprise License installed
- At least one installed and running standalone DKP Essential cluster
- All DKP Essential Clusters are upgraded (see page 1705) to the same DKP version as the DKP Managed Enterprise cluster
- The DKP Essential Cluster you want to convert is self-managed (see page 81)
- The DKP Essential Cluster you want to convert only contains its own Cluster API resources and does not contain Cluster API resources from other clusters

See the Licenses (see page 85) page for information on how you can purchase a DKP Enterprise license.

Attaching DKP Enterprise clusters is not supported.

5.5.5.1.2 Downtime Considerations

Your DKP Essential cluster will not be accessible externally for several minutes during the expansion process. Any configuration of the cluster's `Ingress` that requires `traefik-forward-auth` authentication will be affected.

> Access from within the cluster via Kubernetes service hostname (for example, http://SERVICE_NAME.NAMESPACE:PORT) is not affected.

5.5.5.1.2.1 Affected DKP Services

- `dkp-ceph-cluster-dashboard`
- `grafana-logging`
- `kube-prometheus-stack-alertmanager`
- `kube-prometheus-stack-grafana`
- `kube-prometheus-stack-prometheus`
- `kubernetes-dashboard`
- `traefik-dashboard`

5.5.5.1.2.2 Other Services

To verify if your services are affected by `traefik-forward-auth` 's downtime, run the following command:

```
kubectl get ingress -n NS <your_customized_ingress_name>
```

Look for the `traefik.ingress.kubernetes.io/router.middlewares` field in the output. If this field contains the value `kommander-forwardauth@kubernetescrd` , your service will be affected by the downtime.

5.5.5.1.2.3 Duration

The `traefik-forward-auth` service is affected starting with the `PreAttachmentCleanup` conversion stage, and will run normally again after `ResumeFluxOperations` is completed. Observe the conversion progress (see page 882) to monitor your cluster's current status.

5.5.5.1.3 Next Step

https://d2iq.atlassian.net/wiki/spaces/DENT/pages/174391297/Prerequisites+Prepare+your+Cluster+s+Existing+Configurations (see page 862)

5.5.5.1.4 See Also

Troubleshooting (see page 881)

5.5.5.2 Prerequisites: Prepare your Cluster's Existing Configurations

Enterprise Gov Advanced

Before you convert an *Essential* cluster into a *Managed* cluster, review the following information.

5.5.5.2.1 SSO Configuration

After attachment, the SSO configuration of the Management cluster applies to the Managed (formerly Essential) cluster. Any SSO configuration of the former Essential cluster will be deleted.

- If the **Essential cluster has SSO configured**, but the Management cluster has NO configuration, you can copy your Essential cluster's SSO configuration (`dex-controller` resources) to the Management cluster before conversion.
- If your **Management cluster has SSO configured** and the **Essential cluster has another SSO configuration**, you can choose to keep one, or both. To keep the configuration of your Essential cluster, manually copy the `dex-controller` resources to the Management cluster before conversion. DKP maintains the SSO configuration of your Management cluster automatically, unless you manually delete it.

5.5.5.2.2 Domains and Certificates

Any custom domain or certificate configuration you have set up for your Essential cluster remains functional after you turn it into a Managed cluster.

> ⚠ After conversion, any domain or certificate customizations you would like to apply to your Managed cluster must be done via the `KommanderCluster`. This object is now stored in the Management cluster.

5.5.5.2.3 Next Step

Prerequisites: Delete Gitea (see page 863)

5.5.5.3 Prerequisites: Clone Git Repository from Gitea

After your DKP Essential Cluster is converted into a DKP Enterprise Managed cluster, the old instance of Gitea that is used to host all Git repositories in the DKP Essential Cluster, will not be preserved.

Perform the steps in this page to ensure that you have a local copy of the Management Git Repository in the state it was in prior to undergoing the expansion process.

All DKP Platform Applications will be migrated from the DKP Essential Cluster to the DKP Enterprise Managed Cluster.

1. Prior to turning an DKP Essential cluster to a DKP Enterprise Managed Cluster, clone Gitea using the following command:

```
dkp experimental gitea clone
```

2. Verify that the Git Repository has been successfully cloned to your local environment.

```
cd kommander
git remote -v
# output from git remote -v look like
# origin  https://<YOUR_CLUSTER_INGRESS_HOSTNAME>/dkp/kommander/git/kommander/
kommander (fetch)
# origin  https://<YOUR_CLUSTER_INGRESS_HOSTNAME>/dkp/kommander/git/kommander/
kommander (push)
```

5.5.5.3.1 Next Steps

Back up your Cluster's Applications and Persistent Volumes (see page 863)

5.5.5.4 Back up your Cluster's Applications and Persistent Volumes

Enterprise Gov Advanced

Back up and restore your cluster’s applications before attempting converting your DKP Essential cluster into a DKP Enterprise Managed cluster in the scope of Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

The instructions differ depending on the infrastructure provider of your **DKP Essential cluster**.

For AWS, refer to: Back up an AWS Cluster (see page 864)

For Azure, vSphere, GCP and pre-provisioned environments, refer to: Back up an Azure, vSphere, GCP or Pre-provisioned Cluster (see page 869)

5.5.5.4.1 Back up an AWS Cluster

This section contains the instructions necessary to back up and restore a DKP Essential cluster on the AWS environment in the scope of Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

Here is a summary of the steps:

- Prepare your Cluster for Backup - AWS (see page 864)
- Back up a Cluster - AWS Environment (see page 866)

5.5.5.4.1.1 Prepare your Cluster for Backup - AWS

This section describes how to prepare your cluster on an AWS environment, so it can be backed up before you begin with Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

Prerequisites

- Ensure Velero is installed on your Essential cluster
- Install the Velero CLI (see page 923) (Use at least Velero CLI version 1.10.1)
- Ensure kubectl[593] is installed
- Ensure you have admin rights to the DKP Essential cluster

Prepare your Cluster

> ⚠ Run the following commands in the ***DKP Essential*** **cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 106).

Prepare Velero

Enable the CSI snapshotting plug-in by providing a custom configuration of Velero.

1. Create an Override with the custom configuration:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: velero-overrides
  namespace: kommander
```

593 https://kubernetes.io/docs/tasks/tools/

```
data:
  values.yaml: |
    ---
    configuration:
      features: EnableCSI
    initContainers:
      - name: velero-plugin-for-aws
        image: velero/velero-plugin-for-aws:v1.5.2
        imagePullPolicy: IfNotPresent
        volumeMounts:
          - mountPath: /target
            name: plugins
      - name: velero-plugin-for-csi
        image: velero/velero-plugin-for-csi:v0.4.2
        imagePullPolicy: IfNotPresent
        volumeMounts:
          - mountPath: /target
            name: plugins
EOF
```

2. Update the `AppDeployment` to apply the new configuration:

```
cat << EOF | kubectl -n kommander patch appdeployment velero --type='merge' --
patch-file=/dev/stdin
spec:
  configOverrides:
    name: velero-overrides
EOF
```

3. Verify the configuration has been updated before proceeding with the next section:

```
kubectl -n kommander wait --for=condition=Ready kustomization velero
```

The output should look similar to this:

```
kustomization.kustomize.toolkit.fluxcd.io/velero condition met
```

Prepare the AWS IAM Permission

When creating a cluster on AWS, you provided an additional permission as specified in AWS Cluster IAM Policies, Roles, and Artifacts (see page 1162) .

For the CSI plugin to function correctly, you must update the existing IAM role to include an additional policy.

Add the AmazonEBSCSIDriverPolicy[594] policy to the control plane role `control-plane.cluster-api-provider-aws.sigs.k8s.io`:

```
aws iam attach-role-policy \
  --role-name control-plane.cluster-api-provider-aws.sigs.k8s.io \
  --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy
```

This will allow the EBS CSI driver, a volume manager, to have enough permissions to create volume snapshots.

The default control plane role name is `control-plane.cluster-api-provider-aws.sigs.k8s.io`. If you customized this name when creating the AWS cluster, replace the default control plane role with the name you assigned to it.

Prepare the CSI Configuration

Configure a `VolumeSnapshotClass` object on the cluster, so Velero can create a volume snapshot:

```
cat << EOF | kubectl apply -f -
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: aws
  labels:
    velero.io/csi-volumesnapshot-class: "true"
driver: ebs.csi.aws.com
deletionPolicy: Delete
parameters:
EOF
```

Next Step:

Back up a Cluster - AWS Environment (see page 866)

5.5.5.4.1.2 Back up a Cluster - AWS Environment

With this workflow, you can back up and restore your cluster's applications. The backup contains all Kubernetes objects and the Persistent Volumes of your DKP applications.

594 https://docs.aws.amazon.com/aws-managed-policy/latest/reference/AmazonEBSCSIDriverPolicy.html

When backing up a cluster that runs on a cloud provider, Velero captures the state of your cluster in a snapshot. Review Velero's list of cloud providers for CSI[595] compatibility.

Run the following commands in the ***DKP Essential* cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 106).

Back up Instructions

1. Configure Velero to use CSI snapshotting:

   ```
   velero client config set features=EnableCSI
   ```

2. Create a backup with Velero. Use the following flags to reduce the scope of the backup and only include the applications that are affected during the expansion:

   ```
   velero backup create pre-expansion \
     --include-namespaces="kommander,kommander-default-workspace,kommander-
   flux,kubecost" \
     --include-cluster-resources \
     --wait
   ```

   After completion, the output should look similar to this:

   ```
   Backup request "pre-expansion" submitted successfully.
   Waiting for backup to complete. You may safely press ctrl-c to stop waiting -
   your backup will continue in the background.
   ..............................................................................
   ..............................................................................
   ..............................................................................
   ................
   Backup completed with status: Completed. You may check for more information
   using the commands `velero backup describe pre-expansion` and `velero backup
   logs pre-expansion`.
   ```

Verify the Backup

Review the backup has completed successfully:

```
velero backup describe pre-expansion
```

595 https://velero.io/docs/v1.10/supported-providers/

The following example output will vary depending on your cloud provider. Verify that it shows no errors and the `Phase` is `Completed`:

```
Name:         pre-expansion
Namespace:    kommander
Labels:       velero.io/storage-location=default
Annotations:  velero.io/source-cluster-k8s-gitversion=v1.25.5
              velero.io/source-cluster-k8s-major-version=1
              velero.io/source-cluster-k8s-minor-version=25

Phase:  Completed

Errors:    0
Warnings:  0

Namespaces:
  Included:  kommander, kommander-default-workspace, kommander-flux, kubecost
  Excluded:  <none>

Resources:
  Included:        *
  Excluded:        <none>
  Cluster-scoped:  included

Label selector:  <none>

Storage Location:  default

Velero-Native Snapshot PVs:  auto

TTL:  720h0m0s

CSISnapshotTimeout:  10m0s

Hooks:  <none>

Backup Format Version:  1.1.0

Started:    2023-03-15 10:40:25 -0400 EDT
Completed:  2023-03-15 10:44:39 -0400 EDT

Expiration:  2023-04-14 10:40:24 -0400 EDT

Total items to be backed up:  5188
Items backed up:              5188

Velero-Native Snapshots: <none included>
```

Next Step

Expand your platform. Use the UI or CLI:

UI: Convert an Essential Cluster into a Managed Cluster (see page 874)

CLI: Convert an Essential Cluster into a Managed Cluster (see page 875)

Related Topics

Restore your Backup and Retry the Cluster Expansion (see page 884)

5.5.5.4.2 Back up an Azure, vSphere, GCP or Pre-provisioned Cluster

This section contains the instructions necessary to back up and restore a DKP Essential cluster on Azure, vSphere, GCP or Pre-provisioned environments in the scope of Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

Here is a summary of the steps:

- Prepare your Cluster for Backup - Azure, vSphere, GCP, Pre-provisioned Environments (see page 869)
- Back up a Cluster - Azure, vSphere, GCP, and Pre-provisioned Environments (see page 871)

5.5.5.4.2.1 Prepare your Cluster for Backup - Azure, vSphere, GCP, Pre-provisioned Environments

This section describes how to prepare your cluster on AWS, Azure, vSphere, Google Cloud or pre-provisioned environment, so it can be backed up before you begin with Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

Prerequisites

- Ensure Velero is installed on your Essential cluster
- Install the Velero CLI (see page 923) (Use at least Velero CLI version 1.10.1)
- Ensure kubectl[596] is installed
- Ensure you have admin rights to the DKP Essential cluster

Prepare your Cluster

Run the following commands in the ***DKP Essential* cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 106).

Prepare Velero

Enable `restic` backup capabilities by providing a custom configuration of Velero.

596 https://kubernetes.io/docs/tasks/tools/

1. Create an Override with a custom configuration for Velero. This custom configuration deploys the `node-agent` service, which enables `restic`:

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: velero-overrides
  namespace: kommander
data:
  values.yaml: |
    ---
    deployNodeAgent: true
EOF
```

2. Reference the created Override in Velero's `AppDeployment` to apply the new configuration:

```
cat << EOF | kubectl -n kommander patch appdeployment velero --type='merge' --
patch-file=/dev/stdin
spec:
  configOverrides:
    name: velero-overrides
EOF
```

3. Wait until the `node-agent` has been deployed:

```
until kubectl get daemonset -A | grep -m 1 "node-agent"; do 0.1 ; done
```

The `node-agent` is ready after a similar output appears:

```
kommander               node-agent
3         3         0       3            0           <none>
```

4. Verify the configuration has been updated before proceeding with the next section:

```
kubectl -n kommander wait --for=condition=Ready kustomization velero
```

The output should look similar to this:

```
kustomization.kustomize.toolkit.fluxcd.io/velero condition met
```

Prepare your Pods for Backup

> ⚠ Run the following commands in the ***DKP Essential* cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 106).

Annotate the `gitea` pods to ensure `restic` backs up the Persistent Volumes (PVs) of the pods that will be affected during the expansion process. These volumes contain the Git repository information of your DKP Essential cluster.

```
kubectl -n kommander annotate pod/gitea-0 backup.velero.io/backup-volumes=data
```

```
kubectl -n kommander annotate pod/gitea-postgresql-0 backup.velero.io/backup-
volumes=data
```

Next Step:

Back up a Cluster - Azure, vSphere, GCP, and Pre-provisioned Environments (see page 871)

5.5.5.4.2.2 Back up a Cluster - Azure, vSphere, GCP, and Pre-provisioned Environments

With this workflow, you can back up and restore your cluster's applications. This backup contains Kubernetes objects and the Persistent Volumes (PVs) of Gitea pods. Given that Gitea's PVs store information on your cluster's state, you will be able to Restore your Cluster (see page 885), if required.

> ⚠ Run the following commands in the ***DKP Essential* cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 106).

Back up Instructions

1. Create a backup with Velero. Use the following flags to reduce the scope of the backup and only include the applications that are affected during the expansion:

   ```
   velero backup create pre-expansion \
     --include-namespaces="kommander,kommander-default-workspace,kommander-
   flux,kubecost" \
     --include-cluster-resources \
     --snapshot-volumes=false --wait \
     --namespace kommander
   ```

   After completion, the output should look similar to this:

```
Backup request "pre-expansion" submitted successfully.
Waiting for backup to complete. You may safely press ctrl-c to stop waiting -
your backup will continue in the background.
.....................................................................
.....................................................................
.....................................................................
.................
Backup completed with status: Completed. You may check for more information
using the commands `velero backup describe pre-expansion` and `velero backup
logs pre-expansion`.
```

Verify the Backup

1. Ensure the backup has completed successfully:

   ```
   velero backup describe pre-expansion --namespace kommander
   ```

   The following example output will vary depending on your cloud provider. Verify that it shows no errors and the `Phase` is `Completed`:

   ```
   Name:         pre-expansion
   Namespace:    kommander
   Labels:       velero.io/storage-location=default
   Annotations:  velero.io/source-cluster-k8s-gitversion=v1.25.5
                 velero.io/source-cluster-k8s-major-version=1
                 velero.io/source-cluster-k8s-minor-version=25

   Phase:  Completed

   Errors:    0
   Warnings:  0

   Namespaces:
     Included:  kommander, kommander-default-workspace, kommander-flux, kubecost
     Excluded:  <none>

   Resources:
     Included:        *
     Excluded:        <none>
     Cluster-scoped:  included

   Label selector:  <none>

   Storage Location:  default

   Velero-Native Snapshot PVs:  auto
   ```

```
TTL:  720h0m0s

CSISnapshotTimeout:  10m0s

Hooks:  <none>

Backup Format Version:  1.1.0

Started:    2023-03-15 10:40:25 -0400 EDT
Completed:  2023-03-15 10:44:39 -0400 EDT

Expiration:  2023-04-14 10:40:24 -0400 EDT

Total items to be backed up:  5188
Items backed up:              5188

Velero-Native Snapshots: <none included>
```

2. Ensure that the `PodVolumeBackup` objects have been created:

   ```
   kubectl get podvolumebackups -A
   ```

   The output should look similar to this:

   ```
   NAMESPACE   NAME        STATUS      CREATED   NAMESPACE   POD
   VOLUME   REPOSITORY ID
   UPLOADER TYPE   STORAGE LOCATION   AGE
   kommander   ash-59ntg   Completed   39s       kommander   gitea-postgresql-0
   data     s3:https://a54904d80411e4d64b572b96cb3ddb62-477717230.us-
   west-2.elb.amazonaws.com:8085/dkp-velero/restic/kommander   restic
   default            39s
   kommander   ash-5vsbf   Completed   42s       kommander   gitea-0
   data     s3:https://a54904d80411e4d64b572b96cb3ddb62-477717230.us-
   west-2.elb.amazonaws.com:8085/dkp-velero/restic/kommander   restic
   default            42s
   ```

Next Step

Expand your platform. Use the UI or CLI:

UI: Convert an Essential Cluster into a Managed Cluster (see page 874)

CLI: Convert an Essential Cluster into a Managed Cluster (see page 875)

Related Topics

Restore your Backup and Retry the Cluster Expansion (see page 884)

5.5.5.5 UI: Convert an Essential Cluster into a Managed Cluster

Enterprise Gov Advanced

Instructions on how you can convert an Essential Cluster to a Managed cluster with no Networking Restrictions with the DKP UI.

> ⚠ `Ingress` that contains Traefik-Forward-Authentication (see page 976) configuration will be not available during the expansion process, therefore, your DKP Essential cluster will not be accessible externally for several minutes. Access from within the cluster via Kubernetes service hostname (for example, http://SERVICE_NAME.NAMESPACE:PORT) is not affected.
> See Downtime Considerations (see page 861) for more information.

How to attach an existing cluster that has no additional networking restrictions

Use this option when you want to attach a cluster that does not require additional access information.

1. From the top menu bar, select your target workspace.
2. On the Dashboard page, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
3. Select **Attach Cluster**.
4. Select the **No additional networking restrictions** card. Alternatively, if you MUST use network restrictions, stop following the steps below, and see the instructions on the page Attach a Cluster with Network Restrictions (see page 804).
5. In the **Cluster Configuration** section, paste your `kubeconfig` file into the field, or select the **Upload kubeconfig File** button to specify the file.
6. The **Cluster Name** field will automatically populate with the name of the cluster is in the `kubeconfig`. You can edit this field with the name you want for your cluster.
7. The **Context** select list is populated from the `kubeconfig`. Select the desired context with admin privileges from the **Context** select list.
8. Add labels to classify your cluster as needed.
9. Select **Create** to attach your cluster.

5.5.5.5.1 Verify the Conversion

Run the following commands in the ***Management*** **cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 106).

1. Export the environment variable for the workspace namespace:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

2. To verify that the conversion is successful, check the `KommanderCluster` object:

   ```
   kubectl wait --for=condition=AttachmentCompleted kommandercluster <cluster
   name> -n ${WORKSPACE_NAMESPACE} --timeout 30m
   ```

   The following output appears if the conversion is successful:

   ```
   kommandercluster.kommander.mesosphere.io/<cluster name> condition met
   ```

After conversion, all Platform Applications will be in the Kommander Namespace in the Managed Cluster.

5.5.5.5.2 Next Steps

Post Conversion Cleanup: Cluster Autoscaler Configuration (see page 877)

5.5.5.5.3 See Also

Troubleshooting (see page 881)

5.5.5.6 CLI: Convert an Essential Cluster into a Managed Cluster

Enterprise Gov Advanced

Instructions on how you can convert an DKP Essential Cluster to a DKP Enterprise Managed Cluster with no Networking Restrictions via the CLI.

> ⚠ `Ingress` that contains Traefik-Forward-Authentication (see page 976) configuration will be not available during the expansion process, therefore, your DKP Essential cluster will not be accessible externally for several minutes. Access from within the cluster via Kubernetes service hostname (for example, http://SERVICE_NAME.NAMESPACE:PORT) is not affected.
> See Downtime Considerations (see page 861) for more information.

1. Run the following command in the DKP Enterprise cluster you will have managing your clusters. The cluster name and kubeconfig is from the cluster you are attaching and want to convert to initiate the process of turning an DKP Essential cluster to a DKP Enterprise Managed cluster. The workspace is the workspace name you want the attached cluster to go into.

```
dkp attach cluster --name <essential-cluster-name> --attached-kubeconfig
<kubeconfig-file-of-essential-cluster> --workspace <workspace-name>
```

5.5.5.6.1 Verify the Conversion

> ⚠ Run the following commands in the ***Management*** **cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 106).

1. Export the environment variable for the workspace namespace:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

2. To verify that the conversion is successful, check the `KommanderCluster` object:

```
kubectl wait --for=condition=AttachmentCompleted kommandercluster <cluster
name> -n ${WORKSPACE_NAMESPACE} --timeout 30m
```

   The following output appears if the conversion is successful:

```
kommandercluster.kommander.mesosphere.io/<cluster name> condition met
```

> After conversion, all Platform Applications will be in the `kommander` namespace in the Managed Cluster.

5.5.5.6.2 Next Steps

Post Conversion Cleanup: Cluster Autoscaler Configuration (see page 877)

5.5.5.6.3 See Also

Troubleshooting (see page 881)

5.5.5.7 Post Conversion Cleanup: Cluster Autoscaler Configuration

Follow the steps in this page to ensure that the Cluster Autoscaler is able to function properly after turning your DKP Essential cluster to a DKP Enterprise Managed cluster.

After converting your cluster from Essential to Enterprise, the Cluster Lifecycle Management responsibilities is moved to a single Management cluster.

The Cluster Autoscaler feature also depends on the same Cluster Lifecycle Management components. If you are using the Cluster Autoscaler feature in DKP, you must perform the following steps for this feature to continue to work correctly:

Run the following commands in the ***Management*** **cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 106).

1. Set the following environment variables with your cluster's details:

```
export CLUSTER_NAME=<>
export WORKSPACE_NAMESPACE=<>
```

2. Apply the Cluster Autoscaler `Deployment` and supporting resources:

```
cat <<EOF | kubectl apply -f -
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: cluster-autoscaler-${CLUSTER_NAME}
  name: cluster-autoscaler-${CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cluster-autoscaler-${CLUSTER_NAME}
```

```
  template:
    metadata:
      labels:
        app: cluster-autoscaler-${CLUSTER_NAME}
    spec:
      containers:
      - args:
        - --cloud-provider=clusterapi
        - --node-group-auto-discovery=clusterapi:clusterName=${CLUSTER_NAME}
        - --kubeconfig=/workload-cluster/kubeconfig
        - --clusterapi-cloud-config-authoritative
        - -v5
        command:
        - /cluster-autoscaler
        image: us.gcr.io/k8s-artifacts-prod/autoscaling/cluster-
autoscaler:v1.25.0
        name: cluster-autoscaler
        volumeMounts:
        - mountPath: /workload-cluster
          name: kubeconfig
          readOnly: true
      serviceAccountName: cluster-autoscaler-${CLUSTER_NAME}
      terminationGracePeriodSeconds: 10
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/master
      - effect: NoSchedule
        key: node-role.kubernetes.io/control-plane
      volumes:
      - name: kubeconfig
        secret:
          items:
          - key: value
            path: kubeconfig
          secretName: ${CLUSTER_NAME}-kubeconfig
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-autoscaler-management-${CLUSTER_NAME}
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-autoscaler-management-${CLUSTER_NAME}
subjects:
- kind: ServiceAccount
  name: cluster-autoscaler-${CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
---
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
  name: cluster-autoscaler-${CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-autoscaler-management-${CLUSTER_NAME}
rules:
- apiGroups:
  - cluster.x-k8s.io
  resources:
  - machinedeployments
  - machinedeployments/scale
  - machines
  - machinesets
  verbs:
  - get
  - list
  - update
  - watch
EOF
```

3. Verify the output is similar to the following:

```
deployment.apps/cluster-autoscaler-<cluster-name> created
clusterrolebinding.rbac.authorization.k8s.io/cluster-autoscaler-management-
<cluster-name> created
serviceaccount/cluster-autoscaler-<cluster-name> created
clusterrole.rbac.authorization.k8s.io/cluster-autoscaler-management-<cluster-
name> created
```

4. To check that the status of the deployment has the expected `AVAILABLE` count of `1`, run the following command and verify that the output is similar:

```
$ kubectl get deployment -n $WORKSPACE_NAMESPACE cluster-autoscaler-
$CLUSTER_NAME
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
cluster-autoscaler-<cluster-name>   1/1     1            1           1m
```

5.5.5.7.1 Next Step

Post Conversion Cleanup: Clusters run on Different Cloud Platforms (see page 880)

5.5.5.7.2 See Also

Troubleshooting (see page 881)

5.5.5.8 Post Conversion Cleanup: Clusters run on Different Cloud Platforms

For a DKP Enterprise Management cluster to manage a cluster hosted in another cloud provider, you must ensure the Management cluster has all the necessary permissions.

5.5.5.8.1 Prerequisites

Prior to running these commands, you must ensure that the DKP Management Enterprise cluster is configured with the necessary platform specific permissions to manage the incoming CAPI objects that backs the infrastructure resources in the target cloud platform.

For example, for the DKP Enterprise Managed cluster to manage CAPI clusters in AWS, refer tohttps://cluster-api-aws.sigs.k8s.io/topics/iam-permissions.html.

DKP supports expanding your platform in the following scenarios:

***DKP Enterprise Management cluster* host provider**	***DKP Enterprise Management cluster* IAM permissions**	***DKP Essential cluster* host provider**
AWS	https://cluster-api-aws.sigs.k8s.io/topics/iam-permissions.html	AWS, GCP, vSphere, Pre-provisioned
GCP	https://cloud.google.com/iam/docs/overview	AWS, GCP, vSphere, Pre-provisioned
vSphere	https://docs.vmware.com/en/vRealize-Operations/Cloud/com.vmware.vcom.config.doc/GUID-F85638E3-937E-4E31-90D0-9D4A5E479292.html	AWS, GCP, vSphere, Pre-provisioned
Azure	https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-ops-guide-iam	Azure
Pre-provisioned	NA	AWS, GCP, vSphere, Pre-provisioned

5.5.5.8.2 Moving the CAPI Resources

1. Following the conversion into a DKP Enterprise managed cluster, run the following command to move the CAPI Objects:

```
dkp move capi-resources --from-kubeconfig <essential_cluster_kubeconfig> --to-
kubeconfig <enterprise_cluster_kubeconfig> --to-namespace $
{WORKSPACE_NAMESPACE}
```

2. Verify that the output looks similar to the following:

```
✓ Moving cluster resources
You can now view resources in the moved cluster by using the --kubeconfig flag
with kubectl. For example: kubectl --kubeconfig=<enterprise_cluster_kubeconfig>
get nodes
```

3. After moving the resources, run the following command to remove the CAPI controller manager deployments:

```
dkp delete capi-components --kubeconfig <essential_cluster_kubeconfig>
```

5.5.5.8.3 Next Step

Troubleshooting (see page 881)

5.5.5.9 Troubleshooting

Enterprise Gov Advanced

When an error or failure occurs when converting a DKP Essential cluster to a DKP Enterprise Managed cluster, DKP automatically keeps retrying the cluster's conversion and attachment process. You do not need to trigger it manually.

If the state does not improve after a while, here are some ways in which you can check or troubleshoot the failed conversion:

- Verify the Conversion Status of your Cluster (see page 881)
- Errors Related to CAPI Resources (see page 883)
- Restore your Backup and Retry the Cluster Expansion (see page 884)

5.5.5.9.1 Verify the Conversion Status of your Cluster

Enterprise Gov Advanced

5.5.5.9.1.1 Verify the Conversion

> ⚠ Run the following commands in the ***Management*** **cluster**. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 106).

1. Export the environment variable for the workspace namespace:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

2. To verify that the conversion is successful, check the `KommanderCluster` object:

   ```
   kubectl wait --for=condition=AttachmentCompleted kommandercluster <cluster
   name> -n ${WORKSPACE_NAMESPACE} --timeout 30m
   ```

   The following output appears if the conversion is successful:

   ```
   kommandercluster.kommander.mesosphere.io/<cluster name> condition met
   ```

If the condition is not met yet, you can observe the conversion process:

5.5.5.9.1.2 Observe the Conversion Process

1. Export the environment variable for the workspace namespace:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

2. Print the state of your cluster's `KommanderCluster` object through the CLI, and observe the cluster conversion process:

   ```
   kubectl get kommandercluster -n ${WORKSPACE_NAMESPACE} <essential_cluster_name>
   -o go-template='{{range .status.conditions }}type: {{.type}} {{"\n"}}status:
   {{.status}} {{"\n"}}reason: {{.reason}} {{"\n"}}lastTxTime:
   {{.lastTransitionTime}} {{"\n"}}message: {{.message}} {{"\n\n"}}{{end}}'
   ```

   The output looks similar to this:

   ```
   type: IngressAddressReady
   ```

```
status: False
reason: IngressServiceNotFound
lastTxTime: 2023-02-24T14:58:18Z
message: Ingress service object was not found in the cluster

type: IngressCertificateReady
status: True
reason: Ready
lastTxTime: 2023-02-24T14:49:09Z
message: Certificate is up to date and has not expired

type: CAPIResourceMoved
status: True
reason: Succeeded
lastTxTime: 2023-02-24T14:50:56Z
message: Moved CAPI resources from attached cluster to management cluster

type: PreAttachmentCleanup
status: True
reason: Succeeded
lastTxTime: 2023-02-24T14:54:47Z
message: pre-attach cleanup succeeded

# [...]
```

5.5.5.9.1.3 Next Step:

Errors Related to CAPI Resources (see page 883)

5.5.5.9.2 Errors Related to CAPI Resources

Enterprise Gov Advanced

5.5.5.9.2.1 Failed Condition Reason: `FailedToIdentifyCAPIResources`

kubeconfigRef points to the wrong secret

Verify the `KommanderCluster` object of both your Essential and Enterprise clusters. The `spec.kubeconfigRef.name` of each object should point to a valid `kubeconfig` secret.

1. Download the referenced `kubeconfig` to your local machine:

```
kubectl get secret -n <WORKSPACE_NAMESPACE> <cluster_name>-kubeconfig -o
jsonpath='{.data.value}' | base64 --decode > <cluster_name>-kubeconfig
```

2. Verify if the `kubeconfig` is valid:

```
kubectl get namespaces -A --kubeconfig <cluster_name>-kubeconfig
```

3. Verify the output of the previous command:

No errors in the output

If your output shows no errors, the error message is not related to a `kubeconfig`.

Error in the output

If the output shows an error, delete the `KommanderCluster` object via CLI:

```
kubectl delete kommandercluster -n <WORKSPACE_NAMESPACE> <WRONG_KOMMANDER_CLUSTER>
```

At this particular stage and in the context of converting your cluster, deleting your `KommanderCluster` will not affect your environment. However, DO NOT delete your `KommanderCluster` in other scenarios, as it detaches the referenced cluster from the Management cluster.

Finally, restart the cluster conversion process with the UI (see page 874) or CLI (see page 875).

Essential cluster has more than one instance of `v1beta1/clusters.cluster.x-k8s.io`

> D2iQ does not support converting Essential clusters that contain the Cluster API resources of more than one cluster.

Ensure your Essential cluster only contains its own CAPI resources and does not contain the CAPI resources of other clusters.

5.5.5.9.2.2 Next Topic:

Restore your Backup and Retry the Cluster Expansion (see page 884)

5.5.5.9.3 Restore your Backup and Retry the Cluster Expansion

When an error or failure occurs when converting a DKP Essential cluster to a DKP Enterprise Managed cluster, DKP automatically keeps retrying the cluster's conversion and attachment process. You do not need to trigger it manually.

If you must interrupt the expansion process to restore your cluster and retry the expansion procedure, follow these instructions:

- Restore your Cluster (see page 885)
- Move your CAPI Resources Back to the DKP Essential Cluster (see page 886)

- Verify the Restore Process and Retry the Expansion (see page 886)

5.5.5.9.3.1 Restore your Cluster

Prerequisites

- You have backed up your cluster (see page 863).
- The cluster expansion (see page 859) you attempted was not successful.

Restore the Cluster

Switch between **DKP Enterprise Management** and **DKP Essential** clusters for the following commands. For general guidelines on how to set the context, refer to Provide Context for Commands with a kubeconfig File (see page 106).

1. Delete the `KommanderCluster` object on the **DKP Enterprise Management** cluster:

   ```
   kubectl -n <WORKSPACE_NAMESPACE> delete kommandercluster
   <KOMMANDER_CLUSTER_NAME> --wait=false
   ```

2. Disable the Flux controllers on the **DKP Essential** cluster to interrupt the expansion process:

   ```
   kubectl -n kommander-flux delete deployment -l app.kubernetes.io/
   instance=kommander-flux
   ```

3. Delete the `kube-federation-system` namespace on the **DKP Essential** cluster:

   ```
   kubectl get ns kube-federation-system -o json | jq '.spec.finalizers = []' |
   kubectl replace --raw "/api/v1/namespaces/kube-federation-system/finalize" -f -
   ```

4. Restore your cluster's configuration on the **DKP Essential** cluster:

   ```
   velero restore create pre-expansion --from-backup pre-expansion --existing-
   resource-policy update --wait --namespace kommander
   ```

Next Step:

Move your CAPI Resources Back to the DKP Essential Cluster (see page 886)

5.5.5.9.3.2 Move your CAPI Resources Back to the DKP Essential Cluster

Previous Step:

Restore your Cluster (see page 885)

Move your cluster's CAPI Resources

Because the created backup does not include CAPI resources, you will also have to move the resources back to the DKP Essential cluster.

Ensure you replace `<CAPI_CLUSTER_NAME>` with the name of the DKP Essential cluster you were converting in the current workflow. If you accidentally provide the CAPI cluster name of another Managed cluster, the command will move the CAPI resources of the incorrect cluster to the DKP Essential cluster.

1. Retrieve your Managed cluster's kubeconfig and write it to the `<essential.conf>` file:

```
KUBECONFIG=management.conf ./dkp get kubeconfig -n <WORKSPACE_NAMESPACE> -c
<CAPI_CLUSTER_NAME> > essential.conf
```

2. Move your CAPI resources back to the DKP Essential cluster:

```
dkp move capi-resources --from-kubeconfig management.conf --to-kubeconfig
<essential.conf> -n <WORKSPACE_NAMESPACE> --to-namespace default
```

Next Step:

Verify the Restore Process and Retry the Expansion (see page 886)

5.5.5.9.3.3 Verify the Restore Process and Retry the Expansion

Previous Step:

Move your CAPI Resources Back to the DKP Essential Cluster (see page 886)

Verify the Restore Process

Verify that you successfully restored your cluster:

```
velero restore describe pre-expansion --namespace kommander
```

The output looks similar to this:

```
Name:         pre-expansion
Namespace:    kommander
Labels:       <none>
Annotations:  <none>

Phase:                       Completed
Total items to be restored:  2411
Items restored:              2411

...
```

Retry the Expansion Process

Execute the cluster expansion again, as shown in Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

Related Topic:

Troubleshooting (see page 881)

5.5.6 Advanced Creation of CLI Clusters

Create CLI clusters

This feature is for advanced users and users in unique environments only. We highly recommend using other documented methods to create clusters whenever possible.

5.5.6.1 Generate Cluster Objects

Depending on your infrastructure, DKP CLI can generate a set of cluster objects that can be customized for unusual use cases. Visit the Advanced Configuration documentation for AWS (see page 1190) for an example on how to use the `--output` flags to create a set of cluster objects.

You must set the target `namespace` with the name of the **workspace** you are creating the cluster in using the `dkp create cluster ... --namespace <WORKSPACE_NAME> --dry-run --output=yaml` command. In other words, the `--namespace` flag should equal the **workspace name**.

5.5.6.2 Use the Upload YAML Form

1. In the selected workspace Dashboard, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
2. On the Add Cluster page, select **Upload YAML to Create a Cluster** and provide advanced cluster details:
   - **Workspace**: The workspace where this cluster belongs (if within the Global workspace).
   - **Cluster YAML**: Paste or upload your customized set of cluster objects into this field. Only valid YAML is accepted.
   - **Add Labels**: By default, your cluster has labels that reflect the infrastructure provider provisioning. For example, your AWS cluster may have a label for the data center region and `provider: aws`. Cluster labels are matched to the selectors created for Projects (see page 716). Changing a cluster label may add or remove the cluster from projects.
3. Click **Create** to begin provisioning the DKP CLI cluster. This step may take a few minutes, taking time for the cluster to be ready and fully deploy its components. The cluster automatically tries to join, and should resolve after it is fully-provisioned.

5.5.7 Custom Domains and Certificates Configuration for All Cluster Types

Configure a custom domain and certificate after installing the Kommander component.
For any cluster type, including Managed (see page 80) and Attached (see page 80) clusters.

Configuration Method	*WHILE* installing the Kommander component	*AFTER* installing the Kommander component
Supported cluster types	Only Essential or Management clusters	All cluster types
Documentation	Go to Custom Domains and Certificates for Management and Essential Clusters (see page 1565)	Remain in this section

DKP supports configuring a custom **domain** name for accessing the UI and other platform services, as well as setting up manual or automatic **certificate renewal** or rotation. This section provides instructions and examples on how to configure a customized domain and certificate on **Essential** (see page 81), **Management** (see page 80), **Managed** (see page 80) or **Attached** (see page 80) clusters.

Section Contents:

5.5.7.1 Why Should You Set Up a Custom Domain or Certificate?

5.5.7.1.1 Reasons for Using a Custom DNS Domain

DKP supports the customization of domains to allow you to use your own domain or hostname for your services. For example, you can set up your DKP UI or any of your clusters to be accessible with your custom domain name instead of the domain provided by default.

- To set up a custom domain (without a custom certificate), refer to Configure a Custom Domain without a Custom Certificate (see page 1576).

5.5.7.1.2 Reasons for Using a Custom Certificate

DKP's default CA identity supports the encryption of data exchange and traffic (between your client and your environment's server). To configure an additional security layer that validates your environment's server authenticity, DKP supports configuring a custom certificate issued by a trusted Certificate Authority either directly in a Secret or managed automatically using the ACME protocol (for example, Let's Encrypt).

Changing the default certificate for any of your clusters can be helpful. For example, you can adapt it to classify your DKP UI or any other type of service as trusted (when accessing a service via a browser).

To set up a custom domain and certificate, refer to the following pages respectively:

- Configure a custom domain and certificate as part of the cluster's installation process (see page 1569). This is only possible for your Management/Essential cluster (see page 79).
- Update your cluster's current domain and certificate configuration as part of your cluster's Day 2 operations (see page 891). You can do this for any cluster type (see page 79) in your environment.

Using Let's Encrypt or other public ACME certificate authorities **does not work in air-gapped scenarios**, as these services require connection to the Internet for their setup. For air-gapped environments, you can either use self-signed certificates issued by the cluster (the default configuration), or a certificate created manually using a trusted Certificate Authority.

5.5.7.1.3 Next Topic:

KommanderCluster and Certificate Issuer Concepts (see page 890)

5.5.7.2 KommanderCluster and Certificate Issuer Concepts

5.5.7.2.1 `KommanderCluster` Object

The `KommanderCluster` resource is an object that contains key information for all types of clusters (see page 79) that are part of your environment, such as:

- Cluster access and endpoint information
- Cluster attachment information
- Cluster status and configuration information

5.5.7.2.2 Issuer Objects: `Issuer`, `ClusterIssuer` or `certificateSecret`?

If you use a certificate issued and managed automatically by `cert-manager`, you need an `Issuer` or `ClusterIssuer` that you reference in your `KommanderCluster` resource. The referenced object must contain the information of your certificate provider.

If you want to use a manually-created certificate, you need a `certificateSecret` that you reference in your `KommanderCluster` resource.

5.5.7.2.3 Location of the `KommanderCluster` and Issuer Objects: Management, Managed or Attached cluster?

In the **Management** (see page 80) or **Essential** (see page 81) cluster, both the `KommanderCluster` and *issuer* objects are stored on the same cluster. The *issuer* can be referenced as an `Issuer`, `ClusterIssuer` or `certificateSecret`.

In **Managed** (see page 80) and **Attached** (see page 80) clusters, the `KommanderCluster` object is stored on the Management cluster. The `Issuer`, `ClusterIssuer` or `certificateSecret` is stored on the Managed or Attached cluster.

5.5.7.2.4 HTTP or DNS solver?

When configuring a certificate for your DKP cluster, you can set up an *HTTP solver* or a *DNS solver*. The HTTP protocol exposes your cluster to the public Internet, whereas DNS keeps your traffic hidden. If you use HTTP, your cluster must be publically accessible (via the ingress or load balancer). If you use DNS, this is not a requirement. See Advanced Configuration: ClusterIssuer (see page 1574) for HTTP and DNS configuration options.

If you are enabling a proxied access (see page 846) for a network-restricted cluster (see page 82), this configuration is restricted to DNS.

5.5.7.2.5 Next Step:

Configure Custom Domains or Custom Certificates post Kommander Installation (see page 891)

5.5.7.3 Configure Custom Domains or Custom Certificates post Kommander Installation

This page contains instructions on how to set up custom certificates for any cluster type (see page 79) after installing DKP.

There are two configuration methods:

Configuration Method	*WHILE* installing the Kommander component	*AFTER* installing the Kommander component
Supported cluster types	Only Essential or Management clusters	All cluster types
Documentation	Go to Configure the Kommander Installation with a Custom Domain and Certificate (see page 1569)	Remain in this page

5.5.7.3.1 Configuration Options

After you have installed the Kommander component of DKP, you can configure a custom domain and certificate by modifying the `KommanderCluster` object of your cluster. You have several options to establish a custom domain and certificate.

Choose an ACME-supported Certificate Authority, if you want the `cert-manager` can automatically handle **certificate renewal** and **rotation**.

I want to use an automatically-generated certificate with ACME

5.5.7.3.1.1 Automatically-generated Certificate

Use a certificate that is managed automatically and supported by `cert-manager`:

1. Create an `Issuer` or `ClusterIssuer` with your certificate provider information. Store this object in the cluster where you want to customize the certificate and domain.
   a. See https://d2iq.atlassian.net/wiki/spaces/DENT/pages/137593048/Configuration+Example+with+Let+s+Encrypt (see page 894) if you want to use DKP's default certificate authority.
   b. For an advanced configuration example, see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/175636481/Configure+the+Kommander+Installation+with+a+Custom+Domain+and+Certificate#basic (see page 1569), and select *I want to use an automatically-generated certificate with ACME and require advanced configuration*.
2. Update the `KommanderCluster` by referencing the **name of the created** `Issuer` or `ClusterIssuer` in the `spec.ingress.issuerRef` field.
   Enter the custom domain name in the `spec.ingress.hostname` field:

```
cat <<EOF | kubectl -n <workspace_namespace> --kubeconfig
<management_cluster_kubeconfig> patch \
kommandercluster <cluster_name>  --type='merge' --patch-file=/dev/stdin
spec:
  ingress:
    hostname: <cluster_hostname>
    issuerRef:
      name: <issuer_name>
      kind: Issuer # or ClusterIssuer depending on the issuer config
EOF
```

> **Certificates issued by another Issuer**
>
> You can also configure a certificate issued by another Certificate Authority. In this case, the CA will determine which information to include in the configuration.
> - Refer to https://cert-manager.io/docs/configuration/ for configuration examples.
> - The `ClusterIssuer`'s name **MUST BE** `kommander-acme-issuer`.

I have a manually-generated certificate

5.5.7.3.1.2 Manually-generated Certificate

Use a manually-created certificate that is **customized for your hostname**.

1. Obtain or create a certificate that is customized for your hostname. Store this object in the workspace namespace of the target cluster.
2. Create a secret with the certificate in the cluster's namespace. Give it a name by replacing `<certificate_secret_name>`:

```
kubectl create secret generic -n "${WORKSPACE_NAMESPACE}"
 <certificate_secret_name> \
  --from-file=ca.crt=$CERT_CA_PATH \
  --from-file=tls.crt=$CERT_PATH \
  --from-file=tls.key=$CERT_KEY_PATH \
  --type=kubernetes.io/tls
```

3. Update the `KommanderCluster` by referencing this **secret** in the `spec.ingress.certificateSecretRef` field and provide the custom domain name in the `spec.ingress.hostname`:

```
cat <<EOF | kubectl -n <workspace_namespace> --kubeconfig
<management_cluster_kubeconfig> patch \
kommandercluster <cluster_name>  --type='merge' --patch-file=/dev/stdin
spec:
  ingress:
    hostname: <cluster_hostname>
    certificateSecretRef:
      name: <certificate_secret_name>
EOF
```

For Kommander to access the secret containing the certificate, it must be located in the workspace namespace of the target cluster.

5.5.7.3.1.3 Next Step:

Verify and Troubleshoot Domain and Certificate Customization (see page 895)

5.5.7.3.1.4 Related topics

Why Should You Set Up a Custom Domain or Certificate? (see page 889)

https://d2iq.atlassian.net/wiki/spaces/DENT/pages/137593048/Configuration+Example+with+Let+s+Encrypt (see page 894)

Advanced Configuration: ClusterIssuer (see page 1574)

5.5.7.3.2 Configuration Example with Let's Encrypt

5.5.7.3.2.1 Configure a Custom Certificate with Let’s Encrypt

Let’s Encrypt is one of the Certificate Authorities (CA) supported by `cert-manager`. To set up a Let's Encrypt certificate, create an `Issuer` or `ClusterIssuer` in the target cluster and then reference it in the `issuerRef` field of the `KommanderCluster` resource.

1. Create the Let’s Encrypt ACME cert-manager issuer:

```
cat <<EOF | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: kommander-acme-issuer
spec:
  acme:
    email: <your_email>
    server: https://acme-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      name: kommander-acme-issuer-account
    solvers:
      - dns01:
          route53:
            region: us-east-1
            role: arn:aws:iam::YYYYYYYYYYYY:role/dns-manager
EOF
```

2. Configure the Management cluster to use your `custom-domain.example.com` with a certificate issued by Let’s Encrypt by referencing the created `ClusterIssuer`:

```
cat <<EOF | kubectl -n kommander --kubeconfig <management_cluster_kubeconfig>
patch \ kommandercluster host-cluster  --type='merge' --patch-file=/dev/stdin
spec:
  ingress:
    hostname: custom-domain.example.com
    issuerRef:
      name: custom-acme-issuer
      kind: ClusterIssuer
EOF
```

5.5.7.3.2.2 Next Step:

Verify and Troubleshoot Domain and Certificate Customization (see page 895)

5.5.7.3.2.3 Related Topics

Why Should You Set Up a Custom Domain or Certificate? (see page 889)

Configure Custom Domains or Custom Certificates post Kommander Installation (see page 891)

5.5.7.3.3 Verify and Troubleshoot Domain and Certificate Customization

If you want to ensure the customization for a domain and certificate is completed, or if you want to obtain more information on the status of the customization, display the status information for the `KommanderCluster`. On the Management cluster (see page 80):

1. Inspect the modified `KommanderCluster` object:

   ```
   kubectl describe kommandercluster -n <workspace_name> <cluster_name>
   ```

2. If the ingress is still being provisioned, the output looks similar to this:

   ```
   [...]
   Conditions:
     Last Transition Time:  2022-06-24T07:48:31Z
     Message:               Ingress service object was not found in the cluster
     Reason:                IngressServiceNotFound
     Status:                False
     Type:                  IngressAddressReady
   [...]
   ```

   If the provisioning has been completed, the output looks similar to this:

   ```
   [...]
   Conditions:
     Last Transition Time:  2022-06-28T13:43:33Z
     Message:               Ingress service address has been provisioned
     Reason:                IngressServiceAddressFound
     Status:                True
     Type:                  IngressAddressReady
     Last Transition Time:  2022-06-28T13:42:24Z
     Message:               Certificate is up to date and has not expired
     Reason:                Ready
     Status:                True
     Type:                  IngressCertificateReady
   ```

```
[...]
```

The same command also prints the actual customized values for the `KommanderCluster.Status.Ingress`. Here is an example:

```
[...]
   ingress:
    address: 172.20.255.180
    caBundle: LS0tLS1CRUdJTiBD...<output has been shortened>...DQVRFLS0tLS0K
[...]
```

5.5.7.3.3.1 Related Topics

Why Should You Set Up a Custom Domain or Certificate? (see page 889)

https://d2iq.atlassian.net/wiki/spaces/DENT/pages/137593048/Configuration+Example+with+Let+s+Encrypt (see page 894)

Configure Custom Domains or Custom Certificates post Kommander Installation (see page 891)

5.5.8 Disconnect or Delete Clusters

5.5.8.1 Disconnect or delete a cluster

5.5.8.2 Disconnect vs. Delete

When you attach a cluster to Kommander that was **not created with Kommander**, you can later detach it. This does not alter the running state of the cluster, but simply removes it from the DKP UI. User workloads, platform services, and other Kubernetes resources are not cleaned up at detach.

After successfully detaching the cluster, manually disconnect the attached cluster's Flux installation from the management Git repository. Otherwise, changes to apps in the managed cluster's workspace will still be reflected on the cluster you just detached. Ensure your `dkp` configuration references the target cluster. You can do this by setting the `KUBECONFIG` environment variable to the appropriate kubeconfig file location[597]. An alternative to initializing the `KUBECONFIG` environment variable is to use the `-kubeconfig=cluster_name.conf` flag. Then, run `kubectl -n kommander-flux patch gitrepo management -p '{"spec":{"suspend":true}}' --type merge` to make the cluster's workloads not managed by Kommander, anymore.

597 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

If you **created a managed cluster with Kommander**, you cannot disconnect the cluster, but you can **delete** the cluster. This completely removes the cluster and all of its cloud assets.

We recommend deleting a managed cluster via the DKP UI.

If you delete the Management (Konvoy) cluster, you can not use Kommander to delete any Managed clusters created by Kommander. If you want to delete all clusters, ensure you delete any Managed clusters before finally deleting the Management cluster.

5.5.8.2.1 Statuses

See Statuses (see page 898) for a list of possible states a cluster can have when it is getting disconnected or deleted.

5.5.8.3 Troubleshooting

I cannot detach an attached cluster that is “Pending”

OR

The cluster I deleted via CLI still appears in the UI with “Error” state

Sometimes detaching or deleting a Kubernetes cluster causes that cluster to get stuck in a “Pending” or “Error” state. This can happen because the wrong `kubeconfig` file is used, or the cluster is just not reachable. In order to detach the cluster, so it does not show in the UI, follow these steps:

1. Determine the `KommanderCluster` resource backing the cluster you tried to attach/detach:

   ```
   kubectl -n WORKSPACE_NAMESPACE get kommandercluster
   ```

   Replace `WORKSPACE_NAMESPACE` with the actual current workspace name. You can find this name by going to `https://YOUR_CLUSTER_DOMAIN_OR_IP_ADDRESS/dkp/kommander/dashboard/workspaces` in your browser.

2. Delete the cluster:

   ```
   kubectl -n WORKSPACE_NAMESPACE delete kommandercluster CLUSTER_NAME
   ```

3. If the resource does not go after a short time, remove its finalizers:

   ```
   kubectl -n WORKSPACE_NAMESPACE patch kommandercluster CLUSTER_NAME --type json
   -p '[{"op":"remove", "path":"/metadata/finalizers"}]'
   ```

This removes the cluster from the DKP UI.

5.5.9 Management Cluster

A guide for the Management Cluster and the Management Cluster Workspace

When you install Kommander, the host cluster is attached in the **Management Cluster Workspace** as the **Management** (see page 80) **Cluster**, called **Management Cluster** in the Global workspace dashboard and **Kommander Host** inside the Management Cluster Workspace. This allows the Management Cluster to be included in Projects (see page 716) and enables the management of its Platform Applications (see page 719) from the Management Cluster Workspace.

Do not attach a cluster in the "Management Cluster Workspace" workspace. This workspace is reserved for your Kommander Management cluster only.

5.5.9.1 Editing

As an attached cluster, you can edit the Management Cluster to add or remove Labels, then you can use these labels to include the Management Cluster in Projects inside of the Management Cluster Workspace.

5.5.9.2 Disconnecting

The Management Cluster cannot be disconnected from the GUI like other attached clusters. Because of this, the Management Cluster Workspace cannot be deleted from the GUI as it will always have the Management Cluster inside itself.

5.5.10 Cluster Statuses

A cluster card’s status line displays both the current status and the version of Kubernetes running in the cluster.

The status list includes these values:

Status	Description
Pending	This is the initial state when a cluster is created or connected.
Pending Setup	The cluster has networking restrictions that require additional setup, and is not yet connected or attached.

Status	Description
Loading Data	The cluster has been added to Kommander and we are fetching details about the cluster. This is the status before `Active`.
Active	The cluster is connected to API server.
Provisioning*	The cluster is being created on your cloud provider. This process may take some time.
Provisioned*	The cluster's infrastructure has been created and configured.
Joining	The cluster is being joined to the management cluster for federation.
Joined	The join process is done, and waiting for the first data from the cluster to arrive.
Deleting*	The cluster and its resources are being removed from your cloud provider. This process may take some time.
Error	There has been an error connecting to the cluster or retrieving data from the cluster.
Join Failed	This status can appear when kubefed does not have permission to create entities in the target cluster.
Unjoining	Kubefed is cleaning up after itself, removing all installed resources on the target cluster.
Unjoined	The cluster has been disconnected from the management cluster.
Unjoin Failed	The Unjoin from kubefed failed or there is some other error with deleting or disconnecting.
Unattached*	The cluster was created manually and the infrastructure has been created and configured. However, the cluster is not attached. Review the Manually attach a CLI-created cluster (see page 856) page to resolve this status.

*These statuses only appear on Managed clusters.

5.5.11 Cluster Resources

The Resources graphs on a cluster card show you a cluster's resource requests, limits, and usage. This allows a quick, visual scan of cluster health. Hover over each resource to get specific details for that specific cluster resource.

Resource	Description
CPU Requests	The requested portion of the total allocatable CPU resource for the cluster, measured in number of cores, such 0.5 cores.
CPU Limits	The portion of the total allocatable CPU resource to which the cluster is limited, measured in number of cores, such as 0.5 cores.
CPU Usage	The amount of the allocatable CPU resource being consumed. Cannot be higher than the configured CPU limit. Measured in number of cores, such as 0.5 cores)
Memory Requests	The requested portion of the total allocatable memory resource for the cluster, measured in bytes, such as 64 GiB.
Memory Limits	The portion of the allocatable memory resource to which the cluster is limited, measured in bytes, such as 64 GiB.
Memory Usage	The amount of the allocatable memory resource being consumed. Cannot be higher than the configured memory limit. Measured in bytes, such as 64 GiB.
Disk Requests	The requested portion of the allocatable ephemeral storage resource for the cluster, measured in bytes, such as 64 GiB.
Disk Limits	The portion of the allocatable ephemeral storage resource to which the cluster is limited, measured in bytes, such as 64 GiB.

For more detailed information, see the Kubernetes documentation[598] about resources.

5.5.12 DKP Platform Applications

Platform Applications are applications that DKP provides out-of-the-box with functionality such as observability, cost management, monitoring, logging, making DKP clusters day 2 production-ready right from install.

598 https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

Platform applications are the applications selected by D2iQ from the open source community for use by the DKP platform. You can visit a cluster's detail page to see which platform applications are enabled under the "Platform Applications" section.

Review the Workspace Platform Configuration Requirements (see page 124) to ensure that the attached clusters have sufficient resources. For more information on platform applications and how to customize them, refer to the pages in the Platform Applications (see page 662) section.

5.5.13 Cluster Applications and Statuses

Applications are installed by the management cluster. You can visit a cluster's detail page to see the application dashboards enabled from the deployed applications under the **Application Dashboards** section.

Under the **Applications** section of the cluster's detail page, you can view the workspace applications enabled for the cluster, grouped by category.

In this section, you can also view the current status of the enabled applications on the cluster, on each application card. Hovering on the status displays details about the status of the application.

Cluster applications can have one of the following statuses:

Status	Description
Enabled	The application is enabled, but the status on the cluster is not available.
Pending	The application is waiting to be deployed.
Deploying	The application is currently being deployed to the cluster.
Deployed	The application has successfully been deployed to the cluster.
Deploy Failed	The application failed to deploy to the cluster.

Review the Workspace Platform Application Configuration Requirements (see page 124) to ensure that the attached clusters have sufficient resources. For more information on applications and how to customize them, see Workspace Applications (see page 691).

5.5.13.1 Custom Cluster Application Dashboard Cards

You can add custom application dashboard cards to the cluster detail page's Applications section by creating a `ConfigMap` on the cluster. The `ConfigMap` must have a `kommander.d2iq.io/application` label applied through the CLI, and must contain both `name` and `dashboardLink` data keys to be displayed. Upon creation of the `ConfigMap`, the DKP UI displays a card corresponding to the

data provided in the `ConfigMap`. Custom application cards have a Kubernetes icon, and can link to a service running in the cluster, or use an absolute URL to link to any accessible URL.

5.5.13.1.1 ConfigMap Example

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: "my-app"
  namespace: "app-namespace"
  labels:
    "kommander.d2iq.io/application": "my-app"
data:
  name: "My Application"
  dashboardLink: "/path/to/app"
```

Key	Description	Required
`metadata.labels."kommander.d2iq.io/application"`	The application name (ID).	X
`data.name`	The display name that describes the application and displays on the custom application card in the user interface.	X
`data.dashboardLink`	The link to the application. This can be an absolute link, `https://www.d2iq.com` or a relative link, `/dkp/kommander/dashboard.` If you use a relative link, the link is built using the cluster's path as the base of the URL to the application.	X
`data.docsLink`	Link to documentation about the application. This is displayed on the application card, but omitted if not present.	

Key	Description	Required
`data.category`	Category with which to group the custom application. If not provided, the application is grouped under the category, “None.”	
`data.version`	A version string for the application. If not provided, “N/A” is displayed on the application card in the user interface.	

Use a command similar to this to create a new custom application `ConfigMap`:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: "my-app"
  namespace: "default"
  labels:
    "kommander.d2iq.io/application": "my-app"
data:
  name: "My Application"
  dashboardLink: "/path/to/app"
EOF
```

5.5.14 Kubernetes Cluster Federation (KubeFed)

Kubernetes Cluster Federation (KubeFed)[599] allows you to coordinate the configuration of multiple Kubernetes clusters from a single set of APIs in a hosting cluster. KubeFed aims to provide mechanisms for expressing which clusters should have their configuration managed and what that configuration should be. The mechanisms that KubeFed provides are intentionally low-level, and intended to be foundational for more complex multicluster use cases, such as deploying multi-geo applications and disaster recovery.

DKP uses KubeFed to manage multiple clusters from the management cluster and also to federate various resources. A `KubefedCluster` object is automatically created for each attached cluster and joined to the management cluster. After they are joined, namespaces can be federated to the clusters - this is how you get workspace and project namespaces created on the attached clusters. From here other resources can be federated into those namespaces, such as ConfigMaps, RBAC, and so on.

See these pages for more information:

- https://github.com/kubernetes-sigs/kubefed/blob/master/docs/concepts.md
- https://github.com/kubernetes-sigs/kubefed/blob/master/docs/userguide.md

599 https://github.com/kubernetes-sigs/kubefed

5.6 Backup and Restore

For production clusters, regular maintenance should include routine backup operations to ensure data integrity and reduce the risk of data loss due to unexpected events. Backup operations should include the cluster state, application state, and the running configuration of both stateless and stateful applications in the cluster.

DKP stores all data as CRDs in the Kubernetes API and you can back up and restore it. Choose a procedure depending on your infrastructure provider:

- Configure Velero (see page 904)
- Back up with Velero (see page 923)

5.6.1 Configure Velero

For default installations, DKP deploys Velero[600] integrated with Rook Ceph[601], operating inside the same cluster.

For production use-cases, D2iQ advises providing an *external* storage class to use with Rook Ceph[602]. See the Rook Ceph in DKP (see page 1034) for more information.

5.6.1.1 Cloud Provider Storage

If you do not want to use Rook Ceph for storing Velero backups, you can configure Velero to use cloud provider storage (see page 904).

- Use Velero with AWS (see page 904)
- Use Velero with Azure (see page 910)
- Use Velero with GCP (see page 917)

5.6.1.2 Use Velero with AWS

Configure Velero to use AWS S3 buckets as storage for its backup operations.

5.6.1.2.1 Overview

Follow these procedures to set up Velero with AWS S3:

- Velero with AWS S3 Buckets - Prepare your Environment (see page 905)
- Velero with AWS S3 Buckets - Configure Velero (see page 906)
- Velero with AWS S3 Buckets - Establish a Backup Location (see page 909)

600 https://velero.io/
601 https://rook.io/
602 https://rook.io/

5.6.1.2.2 Velero with AWS S3 Buckets - Prepare your Environment

5.6.1.2.2.1 Prerequisites

- Ensure you have installed Velero (included in the default DKP installation).
- Ensure you have installed the Velero CLI (see page 923).
- You have created an S3 bucket with AWS[603].

5.6.1.2.2.2 Set Environment Variables

- Set the `BUCKET` environment variable to the name of the S3 bucket you want to use as backup storage:

  ```
  export BUCKET=<aws-bucket-name>
  ```

- Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace. Replace `<workspace_namespace>` with the name of the target workspace:

  ```
  export WORKSPACE_NAMESPACE=<workspace_namespace>
  ```

  This can be the `kommander` namespace for the Management (see page 80) cluster or any other additional workspace namespace for Attached (see page 80) or Managed (see page 80) clusters. To list all available workspace namespaces, use the `kubectl get kommandercluster -A` command.

- Set the `CLUSTER_NAME` environment variable. Replace `<target_cluster>` with the name of the cluster where you want to set up Velero:

  ```
  export CLUSTER_NAME=<target_cluster>
  ```

5.6.1.2.2.3 Prepare your AWS Credentials

See the official docs[604] for details on how to use IAM roles instead of static credentials.

1. Create a file containing your static AWS credentials:
   In this example, the file's name is `credentials-velero`.

603 https://docs.aws.amazon.com/AmazonS3/latest/userguide/creating-bucket.html
604 https://github.com/vmware-tanzu/velero-plugin-for-aws

```
cat << EOF > aws-credentials
  [default]
  aws_access_key_id=<REDACTED>
  aws_secret_access_key=<REDACTED>
EOF
```

2. Create a secret on the cluster where you are installing and configuring Velero by referencing the file created in the previous step. This can be the Management (see page 80), a Managed (see page 80) or an Attached (see page 80) cluster:
   In this example, the secret's name is `velero-aws-credentials`.

```
kubectl create secret generic -n ${WORKSPACE_NAMESPACE} velero-aws-credentials
--from-file=aws=aws-credentials --kubeconfig=${CLUSTER_NAME}.conf
```

5.6.1.2.2.4 Next Step:

Velero with AWS S3 Buckets - Configure Velero (see page 906)

5.6.1.2.3 Velero with AWS S3 Buckets - Configure Velero

Customize Velero to allow the configuration of a non-default backup location.

1. Create a `ConfigMap` to enable Velero to use AWS S3 buckets as backup storage location:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: velero-overrides
data:
   values.yaml: |
    configuration:
      backupStorageLocation:
      - bucket: ${BUCKET}
        provider: "aws"
        config:
          region: <AWS_REGION> # such as us-west-2
          s3ForcePathStyle: "false"
          insecureSkipTLSVerify: "false"
          s3Url: ""
          # profile should be set to the AWS profile name mentioned in the
secret
          profile: default
        credentials:
```

```
          # With the proper IAM permissions with access to the S3 bucket,
          # you can attach the EC2 instances using the IAM Role, OR fill in
"existingSecret" OR "secretContents" below.
          #
          # Name of a pre-existing secret (if any) in the Velero namespace
          # that should be used to get IAM account credentials.
          existingSecret: velero-aws-credentials
          # The key must be named "cloud", and the value corresponds to the
entire content of your IAM credentials file.
          # For more information, consult the documentation for the velero
plugin for AWS at:
          # [AWS] https://github.com/vmware-tanzu/velero-plugin-for-aws/blob/
main/README.md
          secretContents:
            # cloud: |
            #   [default]
            #   aws_access_key_id=<REDACTED>
            #   aws_secret_access_key=<REDACTED>
EOF
```

2. Patch the Velero `AppDeployment` to reference the created `ConfigMap` with the Velero overrides:
   a. To update Velero in **all clusters in a workspace**:

      ```
      cat << EOF | kubectl -n ${WORKSPACE_NAMESPACE} patch appdeployment
      velero --type="merge" --patch-file=/dev/stdin
      spec:
        configOverrides:
          name: velero-overrides
      EOF
      ```

   b. To update Velero for a **specific cluster** in a workspace, see Customize an Application per Cluster (see page 685).

3. Check the `ConfigMap` on the `HelmRelease` object:

   ```
   kubectl get hr -n kommander velero -o jsonpath='{.spec.valuesFrom[?
   (@.name=="velero-overrides")]}'
   ```

   The output looks like this if the deployment is successful:

   ```
   {"kind":"ConfigMap","name":"velero-overrides"}
   ```

4. Verify that the Velero pod is running:

   ```
   kubectl get pods -A --kubeconfig=${CLUSTER_NAME}.conf |grep velero
   ```

You can also configure Velero by editing the `kommander.yaml` and rerunning the installation. To follow this *alternative* configuration path, expand the following section:

Alternative Configuration path for Management/Essential clusters

5.6.1.2.3.1 Configure Velero on the Management Cluster

1. Refresh the `kommander.yaml` to add the customization of Velero:
   ⚠️ Before running this command, ensure the `kommander.yaml` is the configuration file you are currently using for your environment. Otherwise, your previous configuration will be lost. ⚠️

   ```
   dkp install kommander -o yaml --init > kommander.yaml
   ```

2. Configure DKP to load the plugins and to include the secret in the `apps.velero` section:
   ℹ️ This process has been tested to work with plugins for AWS v1.1.0 and Azure v1.5.1. More recent versions of these plugins can be used, but have not been tested by D2iQ.

   ```
   ...
     velero:
       values: |
         configuration:
         backupStorageLocation:
           bucket: ${BUCKET}
           config:
             region: <AWS_REGION> # such as us-west-2
             s3ForcePathStyle: "false"
             insecureSkipTLSVerify: "false"
             s3Url: ""
             # profile should be set to the AWS profile name mentioned in the
   secret
             profile: default
         credentials:
           # With the proper IAM permissions with access to the S3 bucket,
           # you can attach the EC2 instances using the IAM Role, OR fill in
   "existingSecret" OR "secretContents" below.
           #
           # Name of a pre-existing secret (if any) in the Velero namespace
           # that should be used to get IAM account credentials.
           existingSecret: velero-aws-credentials
           # The key must be named "cloud", and the value corresponds to the
   entire content of your IAM credentials file.
           # For more information, consult the documentation for the velero plugin
   for AWS at:
           # [AWS] https://github.com/vmware-tanzu/velero-plugin-for-aws/blob/
   main/README.md
           secretContents:
             # cloud: |
   ```

```
            #    [default]
            #    aws_access_key_id=<REDACTED>
            #    aws_secret_access_key=<REDACTED>
...
```

3. Use the modified `kommander.yaml` configuration to install this Velero configuration:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

4. Check the `ConfigMap` on the `HelmRelease` object:

```
kubectl get hr -n kommander velero -o jsonpath='{.spec.valuesFrom[?
(@.name=="velero-overrides")]}'
```

   The output looks like this if the deployment is successful:

```
{"kind":"ConfigMap","name":"velero-overrides"}
```

5. Verify that the Velero pod is running:

```
kubectl get pods -A --kubeconfig=${CLUSTER_NAME}.conf |grep velero
```

5.6.1.2.3.2 Next Step:

Velero with AWS S3 Buckets - Establish a Backup Location (see page 909)

5.6.1.2.4 Velero with AWS S3 Buckets - Establish a Backup Location

5.6.1.2.4.1 Create a Backup Storage Location

1. Create a location for the backup by pointing to an existing S3 bucket:

```
velero backup-location create -n ${WORKSPACE_NAMESPACE} <aws-backup-location-
name> \
  --provider aws \
  --bucket ${BUCKET} \
  --config region=<AWS_REGION> \
  --credential=velero-aws-credentials=aws
```

2. Check that the backup storage location is `Available` and that it references the correct S3 bucket:

```
kubectl get backupstoragelocations -n ${WORKSPACE_NAMESPACE} -oyaml
```

If the BackupStorageLocation is not Available, view any error events by using: `kubectl describe backupstoragelocations -n ${WORKSPACE_NAMESPACE}`

5.6.1.2.4.2 Create a Test Backup

1. Create a test backup that is stored in the location you created in the previous section:

   ```
   velero backup create aws-velero-testbackup -n ${WORKSPACE_NAMESPACE} --
   kubeconfig=${CLUSTER_NAME}.conf --storage-location <aws-backup-location-name>
   --snapshot-volumes=false
   ```

2. View your backup:

   ```
   velero backup describe aws-velero-testbackup
   ```

Next Step:

Back up with Velero (see page 923)

5.6.1.3 Use Velero with Azure

Configure Velero to use Azure Blob Storage as storage for its backup operations.

5.6.1.3.1 Overview

Follow these procedures to set up Velero with Azure Blob Storage:

- Velero with Azure Blob Containers - Prepare your Environment (see page 911)
- Velero with Azure Blob Containers - Configure Velero (see page 913)
- Velero with Azure Blob Containers - Establish a Backup Location (see page 915)

5.6.1.3.2 Velero with Azure Blob Containers - Prepare your Environment

5.6.1.3.2.1 Prerequisites

- Ensure you have installed Velero (included in the default DKP installation).
- Ensure you have installed the Velero CLI (see page 923).
- Ensure you have installed the Azure CLI[605].
- Ensure you have sufficient access rights to the Azure storage environment and blob container you want to use for backup. For more information about data authorization, see the official Azure blob storage documentation[606].

5.6.1.3.2.2 Prepare your Environment

1. Create a container in Azure blob storage[607].
2. Set the `BLOB_CONTAINER` environment variable to the name of the blob container you created to use as backup storage:

   ```
   export BLOB_CONTAINER=<Azure-blob-container-name>
   ```

3. Set up a storage account and resource group[608].
4. Set the `AZURE_BACKUP_RESOURCE_GROUP` variable to the name of the resource group you created:

   ```
   AZURE_BACKUP_RESOURCE_GROUP=<azure-resource-group-name>
   ```

5. Set the `AZURE_STORAGE_ACCOUNT_ID` variable to the unique identifier of the storage account you want to use for the backup:
   See https://learn.microsoft.com/en-us/azure/storage/common/storage-account-get-info?toc=%2Fazure%2Fstorage%2Fblobs%2Ftoc.json&bc=%2Fazure%2Fstorage%2Fblobs%2Fbreadcrumb%2Ftoc.json&tabs=azure-cli#get-the-resource-id-for-a-storage-account to obtain the ID. The output shows the entire location path of the storage account. You only need the last part, or storage account name, to set the variable.

   ```
   AZURE_STORAGE_ACCOUNT_ID=<storage-account-name>
   ```

605 https://learn.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest
606 https://learn.microsoft.com/en-us/azure/storage/common/authorize-data-access?toc=%2Fazure%2Fstorage%2Fblobs%2Ftoc.json&bc=%2Fazure%2Fstorage%2Fblobs%2Fbreadcrumb%2Ftoc.json
607 https://learn.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-portal#create-a-container
608 https://learn.microsoft.com/en-us/azure/storage/common/storage-account-create?tabs=azure-cli#create-a-storage-account-1

6. Set the `AZURE_BACKUP_SUBSCRIPTION_ID` variable to the unique identifier of the subscription you want to use for the backup:
   See https://learn.microsoft.com/en-us/cli/azure/account?view=azure-cli-latest#az-account-list to obtain the ID.

   ```
   AZURE_BACKUP_SUBSCRIPTION_ID=<azure-subscription-id>
   ```

7. Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace.
   Replace `<workspace_namespace>` with the name of the target workspace:

   ```
   export WORKSPACE_NAMESPACE=<workspace_namespace>
   ```

   This can be the `kommander` namespace for the Management (see page 80) cluster or any other additional workspace namespace for Attached (see page 80) or Managed (see page 80) clusters. To list all available workspace namespaces, use the `kubectl get kommandercluster -A` command.

8. Set the `CLUSTER_NAME` environment variable. Replace `<target_cluster>` with the name of the cluster where you want to set up Velero:

   ```
   export CLUSTER_NAME=<target_cluster>
   ```

5.6.1.3.2.3 Prepare your Azure Credentials

See https://learn.microsoft.com/en-us/azure/storage/blobs/authorize-data-operations-portal for more details on authorization.

1. Create a `credentials-velero` file with the information required to create a secret. Use the same credentials that you employed when creating the cluster (see page 0).
   These credentials **should not** be Base64 encoded, because Velero will not read them properly.
   Replace the variables in `<...>` with your environment's information. See your Microsoft Azure account to look up the values.

   ```
   cat << EOF > ./credentials-velero
   AZURE_SUBSCRIPTION_ID=${AZURE_BACKUP_SUBSCRIPTION_ID}
   AZURE_TENANT_ID=<AZURE_TENANT_ID>
   AZURE_CLIENT_ID=<AZURE_CLIENT_ID>
   AZURE_CLIENT_SECRET=<AZURE_CLIENT_SECRET>
   ```

```
AZURE_BACKUP_RESOURCE_GROUP=${AZURE_BACKUP_RESOURCE_GROUP}
AZURE_CLOUD_NAME=AzurePublicCloud
EOF
```

2. Use the `credentials-velero` file to create the secret:

```
kubectl create secret generic -n ${WORKSPACE_NAMESPACE} velero-azure-
credentials --from-file=azure=credentials-velero --kubeconfig=$
{CLUSTER_NAME}.conf
```

5.6.1.3.2.4 Next Step:

Velero with Azure Blob Containers - Configure Velero (see page 913)

5.6.1.3.3 Velero with Azure Blob Containers - Configure Velero

Customize Velero to allow the configuration of a non-default backup location.

1. Create a `ConfigMap` to allow Velero to use Azure blob containers as backup storage location:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: velero-overrides
data:
  values.yaml: |
    initContainers:
      - name: velero-plugin-for-microsoft-azure
        image: velero/velero-plugin-for-microsoft-azure:v1.5.1
        imagePullPolicy: IfNotPresent
        volumeMounts:
          - mountPath: /target
            name: plugins
    credentials:
      extraSecretRef: velero-azure-credentials
EOF
```

2. Patch the Velero `AppDeployment` to reference the created `ConfigMap` with the Velero overrides:
   a. To update Velero in **all clusters in a workspace**:

```
cat << EOF | kubectl -n ${WORKSPACE_NAMESPACE} patch appdeployment
velero --type="merge" --patch-file=/dev/stdin
spec:
  configOverrides:
    name: velero-overrides
EOF
```

b. To update Velero for a **specific cluster** in a workspace, see Customize an Application per Cluster (see page 685).

3. Check the `ConfigMap` on the `HelmRelease` object:

```
kubectl get hr -n kommander velero -o jsonpath='{.spec.valuesFrom[?
(@.name=="velero-overrides")]}'
```

The output looks like this if the deployment is successful:

```
{"kind":"ConfigMap","name":"velero-overrides"}
```

4. Verify that the Velero pod is running:

```
kubectl get pods -A --kubeconfig=${CLUSTER_NAME}.conf |grep velero
```

You can also configure Velero by editing the `kommander.yaml` and rerunning the installation. To follow this *alternative* configuration path, expand the following section:

Alternative Configuration Path for Management/Essential Clusters

1. Refresh the `kommander.yaml` to add the customization of Velero:
⚠️ Before running this command, ensure the `kommander.yaml` is the configuration file you are currently using for your environment. Otherwise, your previous configuration will be lost. ⚠️

```
dkp install kommander -o yaml --init > kommander.yaml
```

2. Configure DKP to load the plugins and to include the secret in the `apps.velero` section:
ℹ️ This process has been tested to work with plugin Azure v1.5.1. More recent versions of these plugins can be used, but have not been tested by D2iQ.

```
...
  velero:
    values: |
      initContainers:
```

```
        - name: velero-plugin-for-microsoft-azure
          image: velero/velero-plugin-for-microsoft-azure:v1.5.1
          imagePullPolicy: IfNotPresent
          volumeMounts:
            - mountPath: /target
              name: plugins
      credentials:
        extraSecretRef: velero-azure-credentials
...
```

3. Use the modified `kommander.yaml` configuration to install this Velero configuration:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

4. Check the `ConfigMap` on the `HelmRelease` object:

```
kubectl get hr -n kommander velero -o jsonpath='{.spec.valuesFrom[?
(@.name=="velero-overrides")]}'
```

   The output looks like this if the deployment is successful:

```
{"kind":"ConfigMap","name":"velero-overrides"}
```

5. Verify that the Velero pod is running:

```
kubectl get pods -A --kubeconfig=${CLUSTER_NAME}.conf |grep velero
```

5.6.1.3.3.1 Next Step:

Velero with Azure Blob Containers - Establish a Backup Location (see page 915)

5.6.1.3.4 Velero with Azure Blob Containers - Establish a Backup Location

5.6.1.3.4.1 Create a Backup Storage Location

1. Create a location for the backup by pointing to an existing Azure container:
   Ensure you set the required environment variables as specified in Velero with Azure Blob Containers - Prepare your Environment (see page 911).
   Replace `<azure-backup-location-name>` with a name for the backup location.

```
velero backup-location create <azure-backup-location-name> -n $
{WORKSPACE_NAMESPACE} \
--provider azure \
--bucket ${BLOB_CONTAINER} \
--config resourceGroup=${AZURE_BACKUP_RESOURCE_GROUP},storageAccount=$
{AZURE_STORAGE_ACCOUNT_ID},subscriptionId=${AZURE_BACKUP_SUBSCRIPTION_ID} \
--credential=velero-azure-credentials=azure --kubeconfig=${CLUSTER_NAME}.conf
```

2. Check that the backup storage location is `Available` and that it references the correct Azure container:

```
kubectl get backupstoragelocations -n ${WORKSPACE_NAMESPACE} -oyaml
```

5.6.1.3.4.2 Create a Test Backup

1. Create a test backup that is stored in the location you created in the previous section:

```
velero backup create azure-velero-testbackup -n ${WORKSPACE_NAMESPACE} \
--kubeconfig=${CLUSTER_NAME}.conf \
--storage-location <azure-backup-location-name> \
--snapshot-volumes=false
```

2. View your backup:

```
velero backup describe azure-velero-testbackup
```

Troubleshooting

1. If your backup wasn't created, Velero may have had an issue installing the plugin. If the plugin was not installed, run this command:

```
velero plugin add velero/velero-plugin-for-microsoft-azure:v1.5.1 -n $
{WORKSPACE_NAMESPACE}
```

2. Confirm your `backupstoragelocation` was configured correctly

```
kubectl get backupstoragelocations -n ${WORKSPACE_NAMESPACE}
```

If your backup storage location is "Available", proceed creating a test backup.

```
NAME              PHASE       LAST VALIDATED   AGE   DEFAULT
<azure-backup-location-name>       Available   38s              60m
```

5.6.1.3.4.3 Next Step:

Back up with Velero (see page 923)

5.6.1.4 Use Velero with GCP

Configure Velero to use Google Cloud Storage Buckets as storage for its backup operations.

5.6.1.4.1 Overview

Follow these procedures to set up Velero with Azure Blob Storage:

- Velero with Google Cloud Storage Buckets - Prepare your Environment (see page 917)
- Velero with Google Cloud Storage Buckets - Configure Velero (see page 919)
- Velero with Google Cloud Storage Buckets - Establish a Backup Location (see page 921)

5.6.1.4.2 Velero with Google Cloud Storage Buckets - Prepare your Environment

5.6.1.4.2.1 Prerequisites

- Ensure you have installed Velero (included in the default DKP installation).
- Ensure you have installed the Velero CLI (see page 923).
- Ensure you have installed the gcloud CLI[609].
- **Optional:** You can install the gsutil CLI[610] (or opt to create buckets through the GCS Console)
- Ensure you have created a GCS bucket[611].
- Ensure you have sufficient access rights to the bucket you want to use for backup. For more information about GCP-related access control, refer to the official documentation of the Google Cloud Storage platform[612].

5.6.1.4.2.2 Set Environment Variables

- Set the `BUCKET` environment variable to the name of the GCS container you want to use as backup storage:

609 https://cloud.google.com/sdk/docs/install
610 https://cloud.google.com/storage/docs/gsutil_install
611 https://cloud.google.com/storage/docs/creating-buckets
612 https://cloud.google.com/storage/docs/access-control

```
export BUCKET=<GCS-bucket-name>
```

- Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace. Replace `<workspace_namespace>` with the name of the target workspace:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

This can be the `kommander` namespace for the Management (see page 80) cluster or any other additional workspace namespace for Attached (see page 80) or Managed (see page 80) clusters. To list all available workspace namespaces, use the `kubectl get kommandercluster -A` command.

- Set the `CLUSTER_NAME` environment variable. Replace `<target_cluster>` with the name of the cluster where you want to set up Velero:

```
export CLUSTER_NAME=<target_cluster>
```

5.6.1.4.2.3 Prepare your Google Credentials

You can store your backups in **Google Cloud Platform/GCS buckets**.

> See https://cloud.google.com/storage/docs/creating-buckets#required-roles for more information on setting up access to your bucket.

1. Create a `credentials-velero` file with the information required to create a secret.
Replace `<service-account-email>` with the email address you used to grant permissions to your bucket. The address usually follows the format `<service-account-user>@<gcp-project>.iam.gserviceaccount.com`.

```
gcloud iam service-accounts keys create credentials-velero \
    --iam-account <service-account-email>
```

2. Use the `credentials-velero` file to create the secret:

```
kubectl create secret generic -n ${WORKSPACE_NAMESPACE} velero-gcp-credentials
--from-file=gcp=credentials-velero --kubeconfig=${CLUSTER_NAME}.conf
```

5.6.1.4.2.4 Next Step:

Velero with Google Cloud Storage Buckets - Configure Velero (see page 919)

5.6.1.4.3 Velero with Google Cloud Storage Buckets - Configure Velero

Customize Velero to allow the configuration of a non-default backup location.

1. Create a `ConfigMap` to allow Velero to use GCS buckets as backup storage location:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: velero-overrides
data:
  values.yaml: |
    initContainers:
      - name: velero-plugin-for-gcp
        image: velero/velero-plugin-for-gcp:v1.5.0
        imagePullPolicy: IfNotPresent
        volumeMounts:
          - mountPath: /target
            name: plugins
    credentials:
      extraSecretRef: velero-gcp-credentials
EOF
```

2. Patch the Velero `AppDeployment` to reference the created `ConfigMap` with the Velero overrides:
   a. To update Velero in **all clusters in a workspace**:

```
cat << EOF | kubectl -n ${WORKSPACE_NAMESPACE} patch appdeployment
velero --type="merge" --patch-file=/dev/stdin
spec:
  configOverrides:
    name: velero-overrides
EOF
```

   b. To update Velero for a **specific cluster** in a workspace, see Customize an Application per Cluster (see page 685).

3. Check the `ConfigMap` on the `HelmRelease` object:

```
kubectl get hr -n kommander velero -o jsonpath='{.spec.valuesFrom[?
(@.name=="velero-overrides")]}'
```

The output looks like this if the deployment is successful:

```
{"kind":"ConfigMap","name":"velero-overrides"}
```

4. Ensure that the Velero pod is running:

```
kubectl get pods -A --kubeconfig=${CLUSTER_NAME}.conf |grep velero
```

You can also configure Velero by editing the `kommander.yaml` and rerunning the installation. To follow this *alternative* configuration path, expand the following section:

Alternative Configuration Path for Management/Essential Clusters

5.6.1.4.3.1 Configure Velero on the Management Cluster

1. Refresh the `kommander.yaml` to add the customization of Velero:
   ⚠️ Before running this command, ensure the `kommander.yaml` is the configuration file you are currently using for your environment. Otherwise, your previous configuration will be lost. ⚠️

```
dkp install kommander -o yaml --init > kommander.yaml
```

2. Configure DKP to load the plugins and to include the secret in the `apps.velero` section:
   ℹ️ This process has been tested to work with plugin GCP v1.5.0. More recent versions of these plugins can be used, but have not been tested by D2iQ.

```
...
  velero:
    values: |
      initContainers:
        - name: velero-plugin-for-gcp
          image: velero/velero-plugin-for-gcp:v1.5.0
          imagePullPolicy: IfNotPresent
          volumeMounts:
            - mountPath: /target
              name: plugins
      credentials:
        extraSecretRef: velero-gcp-credentials
...
```

3. Use the modified `kommander.yaml` configuration to install this Velero configuration:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

4. Check the `ConfigMap` on the `HelmRelease` object:

```
kubectl get hr -n kommander velero -o jsonpath='{.spec.valuesFrom[?
(@.name=="velero-overrides")]}'
```

The output looks like this if the deployment is successful:

```
{"kind":"ConfigMap","name":"velero-overrides"}
```

5. Verify that the Velero pod is running:

```
kubectl get pods -A --kubeconfig=${CLUSTER_NAME}.conf |grep velero
```

5.6.1.4.3.2 Next Step:

Velero with Google Cloud Storage Buckets - Establish a Backup Location (see page 921)

5.6.1.4.4 Velero with Google Cloud Storage Buckets - Establish a Backup Location

5.6.1.4.4.1 Create a Backup Storage Location

1. Create a location for the backup by pointing to an existing GCS bucket:
   Ensure you set the required environment variables as specified in Velero with Google Cloud Storage Buckets - Prepare your Environment (see page 917).
   Replace `<gcp-backup-location-name>` with a name for the backup location.

```
velero backup-location create <gcp-backup-location-name> -n $
{WORKSPACE_NAMESPACE} \
--provider gcp \
--bucket $BUCKET \
--credential=velero-gcp-credentials=gcp
```

2. Check that the backup storage location is `Available` and that it references the correct GCS bucket:

```
kubectl get backupstoragelocations -n ${WORKSPACE_NAMESPACE} -oyaml
```

5.6.1.4.4.2 Create a Test Backup

1. Create a test backup that is stored in the location you created in the previous section:

   ```
   velero backup create gcp-velero-testbackup -n ${WORKSPACE_NAMESPACE} \
   --kubeconfig=${CLUSTER_NAME}.conf \
   --storage-location <gcp-backup-location-name> \
   --snapshot-volumes=false
   ```

2. View your backup:

   ```
   velero backup describe gcp-velero-testbackup
   ```

Troubleshooting

1. If your backup wasn't created, Velero may have had an issue installing the plugin. If the plugin was not installed, run this command:

   ```
   velero plugin add velero/velero-plugin-for-gcp:v1.5.0 -n ${WORKSPACE_NAMESPACE}
   ```

2. Confirm your `backupstoragelocation` was configured correctly

   ```
   kubectl get backupstoragelocations -n ${WORKSPACE_NAMESPACE}
   ```

   If your backup storage location is "Available", proceed creating a test backup.

   ```
   NAME                    PHASE       LAST VALIDATED   AGE   DEFAULT
   <gcp-backup-location-name>       Available   38s              60m
   ```

5.6.1.4.4.3 Next Step:

Back up with Velero (see page 923)

5.6.2 Back up with Velero

DKP provides Velero[613] by default, to support backup and restore operations for your Kubernetes clusters and persistent volumes.

- Install the Velero CLI (see page 923)
- Regular Backup Operations (see page 923)
- Restore a cluster from backup (see page 926)
- Backup Service Diagnostics (see page 928)

5.6.2.1 Install the Velero CLI

Although installing the Velero command-line interface is optional and independent of deploying a DKP cluster, having access to the command-line interface provides several benefits. For example, you can use the Velero command-line interface to back up or restore a cluster on demand, or to modify certain settings without changing the Velero configuration.

- By default, DKP sets up Velero to use Rook Ceph over TLS using a self-signed certificate.
- As a result, when using certain commands, you may be asked to use the `--insecure-skip-tls-verify` flag.

Again, the default setup is not suitable for production use-cases.

See the instructions to install the Velero command-line interface[614] for more information.

In DKP, the Velero platform application is installed in the `kommander` namespace, instead of `velero`. Thus, after installing the CLI, we recommend that you set the Velero CLI `namespace` config option so that subsequent Velero CLI invocations will use the correct namespace:

```
velero client config set namespace=kommander
```

5.6.2.2 Regular Backup Operations

Velero provides the following basic administrative functions to back up production clusters:

- Set a Backup Schedule (see page 0)
- Run On-demand Backups (see page 0)
- Restore from a Backup Archive (see page 926)

613 https://velero.io/
614 https://velero.io/docs/v1.5/basic-install/#install-the-cli

- If you want to back up your cluster in the scope of converting a DKP Essential cluster to a DKP Enterprise Managed cluster (see page 859), see https://d2iq.atlassian.net/wiki/spaces/DENT/pages/198770914/Back+up+your+Cluster+s+Applications+and+Persistent+Volumes (see page 863).
- If you require a custom backup location, see how to create one for AWS (see page 909), Azure (see page 915), or GCP (see page 921).

5.6.2.2.1 Prepare your Environment

Before you modify a schedule or create an on-demand backup, set the following environment variables:

1. Specify the workspace namespace of the cluster for which you want to configure the backup:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

2. Specify the cluster for which you want to create the backup:

```
export CLUSTER_NAME=<target_cluster_name>
```

5.6.2.2.2 Set a Backup Schedule

By default, DKP configures a regular, automatic backup of the cluster's state in Velero. The default settings do the following:

- Create daily backups
- Save the data from all namespaces

DKP default backups do not support the creation of Volume Snapshots.

These default settings take effect after the cluster is created. If you install DKP with the default platform services deployed, the initial backup starts after the cluster is successfully provisioned and ready for use.

5.6.2.2.2.1 Create Backup Schedules

The Velero CLI provides an easy way to create alternate backup schedules. For example:

```
velero create schedule <backup-schedule-name> -n ${WORKSPACE_NAMESPACE} \
```

```
--kubeconfig=${CLUSTER_NAME}.conf \
--snapshot-volumes=false \
--schedule="@every 8h"
```

5.6.2.2.2.2 Change Default Backup Service Settings

1. Check the backup schedules currently configured for the cluster:

   ```
   velero get schedules
   ```

2. Delete the `velero-default` schedule:

   ```
   velero delete schedule velero-default
   ```

3. Replace the default schedule with your custom settings:

   ```
   velero create schedule velero-default -n ${WORKSPACE_NAMESPACE} \
   --kubeconfig=${CLUSTER_NAME}.conf \
   --snapshot-volumes=false \
   --schedule="@every 24h"
   ```

5.6.2.2.2.3 Create Backup Schedule for a Specific Namespace

You can also create backup schedules for specific namespaces. Creating a backup for a specific namespace can be useful for clusters running multiple apps operated by multiple teams. For example:

```
velero create schedule <backup-schedule-name> \
--include-namespaces=kube-system,kube-public,kommander \
--snapshot-volumes=false \
--schedule="@every 24h"
```

The Velero command line interface provides many more options worth exploring. You can also find tutorials for disaster recovery[615] and cluster migration[616] on the Velero community site.

5.6.2.2.3 Back up on Demand

In some cases, you might find it necessary to create a backup outside the regularly-scheduled interval. For example, if you are preparing to upgrade any components or modify your cluster configuration, perform a backup before taking that action.

Create a backup by running the following command:

615 https://velero.io/docs/v0.11.0/disaster-case
616 https://velero.io/docs/v0.11.0/migration-case

```
velero backup create <backup-name> -n ${WORKSPACE_NAMESPACE} \
--kubeconfig=${CLUSTER_NAME}.conf \
--snapshot-volumes=false
```

5.6.2.3 Restore a cluster from backup

5.6.2.3.1 Prerequisites

Before attempting to restore the cluster state using the Velero command-line interface, verify the following requirements:

- The backend storage, Rook Ceph Cluster, is still operational.
- The Velero platform service in the cluster is still operational.
- The Velero platform service is set to a `restore-only-mode` to avoid having backups run while restoring.

5.6.2.3.2 Restoring from Backup

To list the available backup archives for your cluster, run the following command:

```
velero backup get
```

Check your deployment to verify that the configuration change was applied correctly:

```
helm get values -n kommander velero
```

If restoring your cluster from a backup, use Read Only Backup Storage. To restore cluster data on demand from a selected backup snapshot available in the cluster, run a command similar to the following:

```
velero restore create --from-backup <BACKUP-NAME>
```

Important: If you are restoring using Velero from the default setup (and not using an external bucket or blob to store your backups (see page 904)), you may see an error when describing or viewing the logs of your backup restore. This is a known issue when restoring from an object store that is not accessible from outside your cluster. However, you can review the success of the backup restore by confirming the Phase is `Completed` and not in error, and viewing the logs by running these `kubectl` commands:

```
kubectl logs -l name=velero -n kommander --tail -1
```

5.6.2.3.3 Restore your DKP Management Cluster

When restoring a backup to the Management Cluster, you must adjust configuration to avoid restore errors.

5.6.2.3.3.1 Prior to restoring a backup

1. Ensure the following `ResourceQuota` setup is not configured on your cluster (this `ResourceQuota` will be automatically restored)

   ```
   kubectl -n kommander delete resourcequota one-kommandercluster-per-kommander-workspace
   ```

2. Turn off the `Workspace` validation webhooks. Otherwise, you will not restore Workspaces with pre-configured namespaces. If the validation webhook named `kommander-validating` is present, it must be modified with this command:

   ```
   kubectl patch validatingwebhookconfigurations kommander-validating \
     --type json \
     --patch '[
       {
         "op": "remove",
         "path": "/webhooks/0/rules/3/operations/0"
       }
     ]'
   ```

5.6.2.3.3.2 After restoring a backup

1. Verify that the `ResourceQuota` named `one-kommandercluster-per-kommander-workspace` is restored
2. Add the removed `CREATE` webhook rule operation with:

   ```
   kubectl patch validatingwebhookconfigurations kommander-validating \
     --type json \
     --patch '[
       {
   ```

```
      "op": "add",
      "path": "/webhooks/0/rules/3/operations/0",
      "value": "CREATE"
    }
  ]'
```

5.6.2.4 Backup Service Diagnostics

You can check whether the Velero service is currently running on your cluster through the Kubernetes dashboard (accessible through the DKP UI on the Management Cluster), or by running the following `kubectl` command:

```
kubectl get all -A | grep velero
```

If the Velero platform service application is currently running, you can generate diagnostic information about Velero backup and restore operations. For example, you can run the following commands to retrieve, back up, and restore information that you can use to assess the overall health of Velero in your cluster:

```
velero get schedules
velero get backups
velero get restores
velero get backup-locations
velero get snapshot-locations
```

5.7 Logging

D2iQ Kubernetes Platform (DKP) comes with a pre-configured logging stack that allows you to collect and visualize pod and admin log data at the Workspace level. The logging stack is also multi-tenant capable. Multi-tenancy is enabled at the Project level through role-based access control (RBAC).

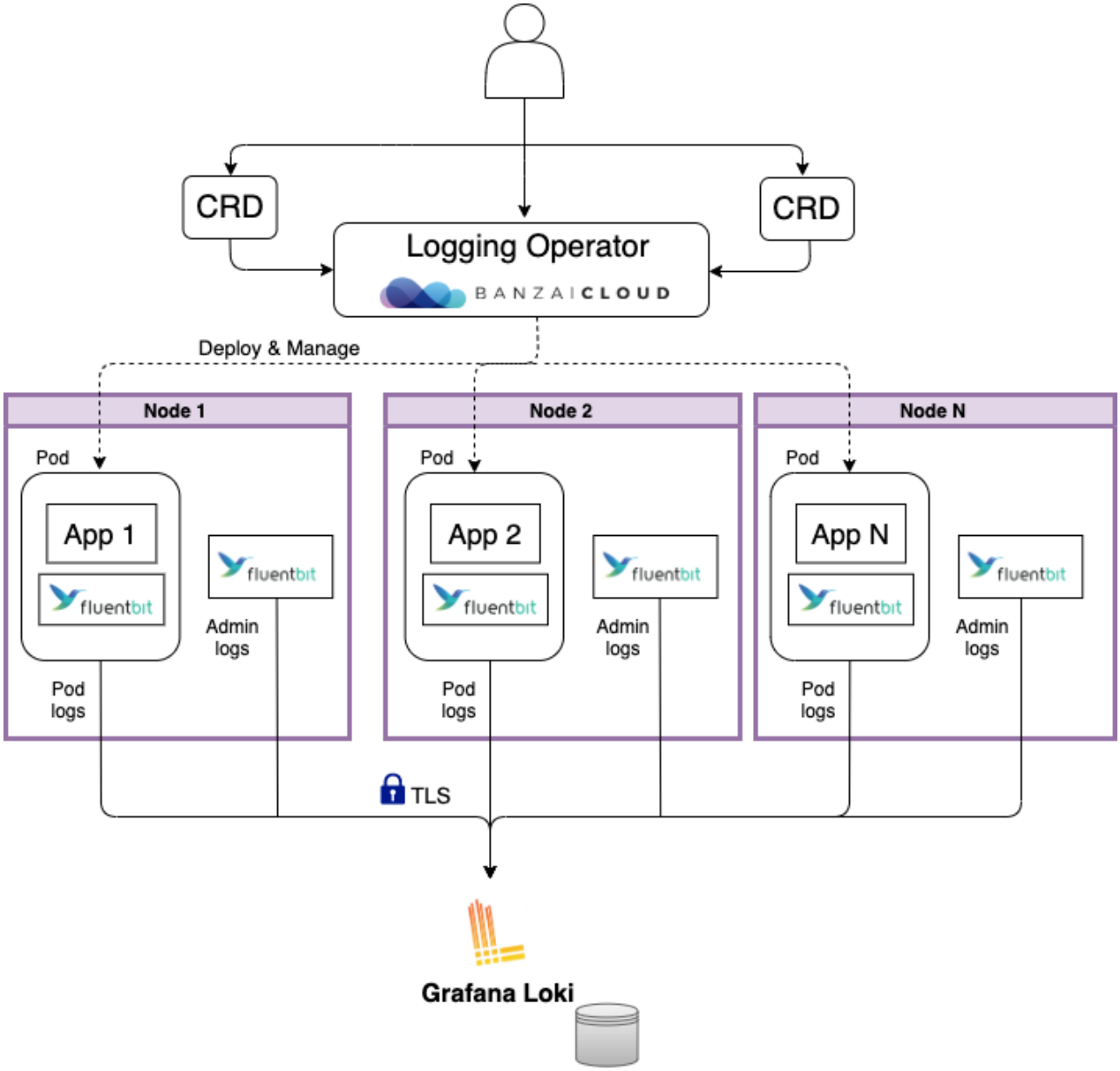
By default, logging is disabled on managed and attached clusters. You need to enable the logging stack applications explicitly on the workspace to make use of these capabilities.

The primary components of the logging stack include these platform services:

- BanzaiCloud Logging-operator
- Grafana and Grafana Loki
- Fluent Bit and Fluentd

In addition to these platform services, logging relies on other software and system facilities, including the container runtime, the journald facility, and systemd configuration are used to collect logs and messages from all the machines in the cluster.

The following diagram illustrates how different components of the logging stack collect log data and provide information about clusters:



The DKP logging stack aggregates logs from applications and nodes running inside your cluster.

DKP uses the BanzaiCloud Logging-operator to manage the Fluent Bit and Fluentd deployments that collect pod logs, using Kubernetes API extensions called custom resources. The custom resources allow users to declare logging configurations using `kubectl` commands. The Fluent Bit instance deployed by Logging-operator gathers pod logs data and sends it to Fluentd, which forwards it to the appropriate Grafana Loki servers based on the configuration defined in custom resources.

Loki then indexes the log data by label and stores it for querying. Loki maintains log order integrity but does not index the log messages themselves, which improves its efficiency and lowers its footprint.

5.7.1 Logging Operator

This section contains information about setting up the logging operator to manage the Fluent Bit and Fluentd resources.

5.7.1.1 Fluent Bit Buffer Information

Fluent Bit collects container logs from the host filesystem and performs the following:

- Maintains a small buffer of logs (5 MB in memory)
- Maintains a checkpoint on each host for each file it's consumed, so if it's restarted, it can resumed where it left off.
- Flushes the buffer every one second.
- Has a five second grace period to flush the buffer on exit (along with a 30 second termination grace period on the pod).

Every pod restart should result in the buffer being flushed to `fluentd` (allows up to five seconds for that flush to happen).

If the buffer is not fully flushed during that five seconds, a small amount of log data may be dropped, but if `fluentd` is functional, it is unlikely that fluent-bit would be unable to flush its logs on pod termination.

Fluent Bit can be configured to use a `hostPath` volume to store the buffer information, so it can be picked up again when Fluent Bit restarts.

For more information about Fluent Bit, see Fluent Bit log collector[617].

For more information about Logging in relation to how it is used in DKP, refer to these pages in our Help Center:

- Admin-level logs (see page 930)
- Enable Workspace-level Logging (see page 931)
- Multi-Tenant Logging Overview (see page 942)
- Fluent Bit (see page 952)
- Scaling the Logging Stack (see page 957)
- Configuring Loki to use AWS S3 Storage in DKP (see page 961)
- Customizing Logging Stack Applications (see page 962)

5.7.2 Admin-level logs

DKP also includes a Fluentbit instance to collect admin-level log information which is sent to the workspace Grafana Loki that's running on the cluster. The admin log information includes:

- Logs for host processes managed by systemd
- Kernel logs
- Kubernetes audit logs

617 https://kube-logging.dev/docs/logging-infrastructure/fluentbit/#hostpath-volumes-for-buffers-and-positions

This approach helps to isolate the more sensitive logs from Logging-operator, eliminating the possibility that users might gain inadvertent access to that data.

See Fluent Bit (see page 952) for more information about these logs.

> ⚠ On the Management cluster, the Fluentbit application is disabled by default. The amount of admin logs ingested to Loki requires additional disk space to be configured on the `rook-ceph-cluster`. Enabling admin logs may use around 2GB/day per node. See Rook Ceph in DKP (see page 1034) for more details on how to configure the Ceph Cluster.

5.7.3 Enable Workspace-level Logging

5.7.3.1 How to enable Workspace-level Logging for use with DKP.

Logging is disabled by default on managed and attached clusters. You will need to enable logging features explicitly at the Workspace level, if you want to capture and view log data.

> ⚠ You must perform these procedures to enable multi-tenant logging at the Project level as well.

- Logging Prerequisites and Architecture (see page 931)
- Enable Logging Applications through the UI (see page 933)
- Create AppDeployments to Enable Workspace Logging (see page 934)
- Override ConfigMap to Restrict Logging to Specific Namespaces (see page 935)
- Override ConfigMap to Modify the Storage Retention Period in Workspace Grafana Loki (see page 937)
- Verify Cluster Logging Stack Installation (see page 939)
- View Cluster Log Data (see page 940)

5.7.3.2 Logging Prerequisites and Architecture

Before you begin, you must:

- Be a cluster administrator with permissions to configure cluster-level platform services.
- Set a default storage class (see page 1531) on each attached cluster for successful Loki deployment.

For more information, see Default Storage Providers in DKP (see page 110) .

5.7.3.2.1 DKP Logging Stack Architecture

The DKP logging stack architecture provides a comprehensive logging solution for the DKP platform. It combines Fluent Bit, Fluentd, Loki, and Grafana components to collect, process, store, and visualize log data. The architecture establishes a robust logging solution by assigning specific roles to each of those components.

5.7.3.2.1.1 Components

1. **Fluent Bit** - Fluent Bit is a lightweight log processor and forwarder that collects log data from various sources, such as application logs or Kubernetes components. It forwards the collected logs to Fluentd for further processing.
2. **Fluentd** - Fluentd is a powerful and flexible log aggregator that receives log data from Fluent Bit, processes it, and forwards it to the Loki Distributor. Fluentd can handle various log formats and enrich the log data with additional metadata before forwarding it.
3. **Loki** - Loki is a horizontally-scalable, highly-available, multi-tenant log aggregation system. Loki components include:- Compactor: Responsible for compacting index files and chunks to improve query performance and reduce storage usage.
    a. Distributor: Receives log streams, partitions them into chunks, and forwards these chunks to the Loki Ingester component.
    b. Gateway: Acts as a single access point to various Loki components, routing requests between Distributor, Query Frontend, and other components as needed.
    c. Ingester: Compresses, indexes, and persists received log chunks.
    d. Querier: Fetches log chunks from the Ingester, decompresses and filters them based on the query, and returns the results to the Query Frontend.
    e. Query Frontend: Splits incoming queries into smaller parts, forwards these to the Loki Querier component, and combines the results from all Queriers before returning the final result.
4. **Grafana** - Grafana is a visualization and analytics platform that supports Loki as one of its data sources. Grafana provides a user-friendly interface for querying and visualizing log data based on user-defined dashboards and panels.

5.7.3.2.1.2 Workflow

- **Write Path**:
    - Fluent Bit instances running on each node collect log data from various sources, like application logs or Kubernetes components.
    - Fluent Bit forwards the collected log data to the Fluentd instance.
    - Fluentd processes the received log data and forwards it to the Loki Distributor through the Loki Gateway.
    - The Loki Distributor receives the log streams, partitions them into chunks, and forwards these chunks to the Loki Ingester component.

    - Loki Ingesters are responsible for compressing, indexing, and persisting the received log chunks.
- **Read Path**:
    - When a user queries logs through Grafana, the request goes to the Loki Gateway, which routes it to the Loki Query Frontend.
    - The Query Frontend splits the query into smaller parts and forwards these to the Loki Querier component.
    - Loki Queriers fetch the log chunks from the Loki Ingester, decompress and filter them based on the query, and return the results to the Query Frontend.
    - The Query Frontend combines the results from all Queriers and returns the final result to Grafana through the Loki Gateway.
    - Grafana visualizes the log data based on the user's dashboard configuration

5.7.3.3 Enable Logging Applications through the UI

5.7.3.3.1 How to enable the logging stack through the UI for Workspace-level logging

You can enable the Workspace logging stack to all attached clusters within the Workspace through the UI. If you prefer to enable the logging stack with `kubectl`, review how you create AppDeployments to Enable Workspace Logging (see page 934).

To enable workspace-level logging in DKP using the UI, follow these steps:

1. From the top menu bar, select your target workspace.
2. Select **Applications** from the sidebar menu.
3. Ensure `traefik` and `cert-manager` are enabled on your cluster. These are deployed by default unless you modified your configuration.
4. Scroll to the **Logging** applications section.
5. Select the three dot button from the bottom-right corner of the cards for Rook Ceph and Rook Ceph Cluster, then click **Enable**. On the **Enable Workspace Platform Application** page, you can add a customized configuration for settings that best fit your organization. You can leave the configuration settings unchanged to enable with default settings.
6. Select **Enable** at the top right of the page.
7. Repeat the process for the Grafana Loki, Logging Operator, and Grafana Logging applications.
8. You can verify the cluster logging stack installation by waiting until the cards have a Deployed checkmark on the Cluster Application (see page 901) page, or you can verify the Cluster Logging Stack installation via the CLI (see page 939).
9. Then, you can view cluster log data (see page 940).

We do not recommend installing Fluent Bit, which is responsible for collecting admin logs, unless you have configured the Grafana Loki Ceph Cluster Bucket with sufficient storage space. The amount of admin logs ingested to Loki requires additional disk space to be configured on the `rook-ceph-cluster`. Enabling admin logs may use around 2GB/day per node. See Rook Ceph in DKP (see page 1034) for more details on how to configure the Ceph Cluster.

5.7.3.4 Create AppDeployments to Enable Workspace Logging

How to create AppDeployments to enable Workspace-level logging

Workspace logging AppDeployments enable and deploy the logging stack to all attached clusters within the workspace. Use the DKP UI to enable the logging applications, or, alternately, use the CLI to create the AppDeployments.

To enable logging in DKP using the CLI, follow these steps on the **management** cluster:

1. Execute the following command to get the name and namespace of your workspace:

   ```
   dkp get workspaces
   ```

   And copy the values under the `NAME` and `NAMESPACE` columns for your workspace.

2. Export the `WORKSPACE_NAME` variable:

   ```
   export WORKSPACE_NAME=<WORKSPACE_NAME>
   ```

3. Export the `WORKSPACE_NAMESPACE` variable:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

4. Ensure that Cert-Manager and Traefik are enabled in the workspace. If you want to find if the applications are enabled on the management cluster workspace, you can run:

   ```
   dkp get appdeployments --workspace ${WORKSPACE_NAME}
   ```

5. You can confirm that the applications are deployed on the **managed** or **attached** cluster by running this `kubectl` command in that cluster. (Ensure you switch to the correct context or kubeconfig[618] of the attached cluster for the following `kubectl` command.)

618 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

```
kubectl get helmreleases -n ${WORKSPACE_NAMESPACE}
```

6. Copy these commands and execute them on the **management** cluster from a command line to create the Logging-operator, Grafana-loki, and Grafana-logging AppDeployments:

```
dkp create appdeployment logging-operator --app logging-operator-3.17.9 --
workspace ${WORKSPACE_NAME}
dkp create appdeployment rook-ceph --app rook-ceph-1.10.3 --workspace $
{WORKSPACE_NAME}
dkp create appdeployment rook-ceph-cluster --app rook-ceph-cluster-1.10.3 --
workspace ${WORKSPACE_NAME}
dkp create appdeployment grafana-loki --app grafana-loki-0.69.16 --workspace $
{WORKSPACE_NAME}
dkp create appdeployment grafana-logging --app grafana-logging-6.57.4 --
workspace ${WORKSPACE_NAME}
```

Then, you can verify the cluster logging stack installation (see page 939).

To deploy the applications to selected clusters within the workspace, refer to the cluster-scoped configuration (see page 682) section.

We do not recommend installing Fluent Bit, which is responsible for collecting admin logs, unless you have configured the Rook Ceph Cluster with sufficient storage space. Enabling admin logs via Fluent Bit may use around 2GB/day per node. See Rook Ceph in DKP (see page 1034) for more details on how to configure the Rook Ceph Cluster.

To install Fluent Bit, create the AppDeployment:

```
dkp create appdeployment fluent-bit --app fluent-bit-0.20.9 --workspace $
{WORKSPACE_NAME}
```

5.7.3.5 Override ConfigMap to Restrict Logging to Specific Namespaces

How to override the logging configMap to restrict logging to specific namespaces.

As a cluster administrator, you may have a need to limit, or restrict, logging activities only to certain namespaces. Kommander allows you to do this by creating an override configMap that modifies the logging configuration created in the Create AppDeployment for Workspace Logging (see page 934) procedure.

5.7.3.5.1 Prerequisites

- Implement each of the steps listed in Enable Workspace-level Logging (see page 931).
- Ensure that log data is available before you execute this procedure.

5.7.3.5.2 Create and use the Override Entries

To create and use the override configMap entries, follow these steps:

1. Execute the following command to get the namespace of your workspace:

   ```
   dkp get workspaces
   ```

   And copy the value under the `NAMESPACE` column for your workspace.

2. Set the `WORKSPACE_NAMESPACE` variable to the namespace copied in the previous step:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

3. Identify one or more namespaces to which you want to restrict logging.

4. Create a file named `logging-operator-logging-overrides.yaml` and paste the following YAML code into it to create the overrides configMap:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: logging-operator-logging-overrides
     namespace: ${WORKSPACE_NAMESPACE}
   data:
     values.yaml: |
       ---
       clusterFlows:
       - name: cluster-containers
         spec:
           globalOutputRefs:
           - loki
           match:
           - exclude:
               namespaces:
               - <your-namespace>
               - <your-other-namespace>
   ```

5. Add the relevant namespace values for `metadata.namespace` and the `clusterFlows[0].spec.match[0].exclude.namespaces` values at the end of the file, and save the file.
6. Use the following command to apply the YAML file:

   ```
   kubectl apply -f logging-operator-logging-overrides.yaml
   ```

7. Edit the `logging-operator` AppDeployment to set the value of `spec.configOverrides.name` to `logging-operator-logging-overrides`.
   (Refer to Deploy an application with a custom configuration (see page 0) for more information)

   ```
   dkp edit appdeployment -n ${WORKSPACE_NAMESPACE} logging-operator
   ```

   After your editing is complete, the AppDeployment resembles this example:

   ```
   apiVersion: apps.kommander.d2iq.io/v1alpha3
   kind: AppDeployment
   metadata:
     name: logging-operator
     namespace: ${WORKSPACE_NAMESPACE}
   spec:
     appRef:
       name: logging-operator-3.17.7
       kind: ClusterApp
     configOverrides:
       name: logging-operator-logging-overrides
   ```

8. Perform actions that generate log data, both in the specified namespaces *and* the namespaces you mean to exclude.
9. Verify that the log data contains only the data you expected to receive.

5.7.3.6 Override ConfigMap to Modify the Storage Retention Period in Workspace Grafana Loki

DKP configures Grafana Loki to retain log metadata and logs for 1 week, using the Compactor[619]. This retention policy can be customized.

The minimum retention period is 24h.

619 https://grafana.com/docs/loki/latest/operations/storage/boltdb-shipper/#compactor

To customize the retention policy using `configOverrides`, run these commands on the **management** cluster:

1. Execute the following command to get the namespace of your workspace:

   ```
   dkp get workspaces
   ```

   Copy the value under the `NAMESPACE` column for your workspace.

2. Set the `WORKSPACE_NAMESPACE` variable to the namespace copied in the previous step:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

3. Create a `ConfigMap` with custom configuration values for Grafana Loki. Since the retention configuration is nested in a `config` string, you must copy the entire block. The following example sets the retention period to 360 hours (15 days). Refer to Grafana Loki's Retention Configuration documentation[620] for more information about this field.

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: grafana-loki-custom-overrides
     namespace: ${WORKSPACE_NAMESPACE}
   data:
     values.yaml: |
       loki:
         structuredConfig:
           limits_config:
             retention_period: 360h
   EOF
   ```

4. Edit the `grafana-loki` AppDeployment to set the value of `spec.configOverrides.name` to `grafana-loki-custom-overrides` (Refer to Deploy a service with a custom configuration (see page 0) for more information).

   ```
   dkp edit appdeployment -n ${WORKSPACE_NAMESPACE} grafana-loki
   ```

   After your editing is complete, the AppDeployment resembles this example:

   ```
   apiVersion: apps.kommander.d2iq.io/v1alpha3
   ```

620 https://grafana.com/docs/loki/latest/operations/storage/retention/#retention-configuration

```
kind: AppDeployment
metadata:
  name: grafana-loki
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: grafana-loki-0.69.10
    kind: ClusterApp
  configOverrides:
    name: grafana-loki-custom-overrides
```

5.7.3.7 Verify Cluster Logging Stack Installation

How to verify the cluster's logging stack installed successfully.

You must wait for the cluster's logging stack `HelmReleases` to deploy before attempting to configure or use the logging features.

Run the following commands on the **management** cluster:

1. Execute the following command to get the namespace of your workspace:

   ```
   dkp get workspaces
   ```

   And copy the value under the `NAMESPACE` column for your workspace.

2. Set the `WORKSPACE_NAMESPACE` variable to the namespace copied in the previous step:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

   Ensure you switched to the correct context or kubeconfig[621] of the attached cluster for the following kubectl commands.

3. Check the deployment status using this command on the **attached** cluster:

   ```
   kubectl get helmreleases -n ${WORKSPACE_NAMESPACE}
   ```

NOTE: It may take some time for these changes to take effect, based on the duration configured for the Flux GitRepository reconciliation.

621 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

When the logging stack is successfully deployed, you will see output that includes the following `HelmReleases`:

```
NAME                          READY   STATUS                             AGE
grafana-logging               True    Release reconciliation succeeded   15m
logging-operator              True    Release reconciliation succeeded   15m
logging-operator-logging      True    Release reconciliation succeeded   15m
grafana-loki                  True    Release reconciliation succeeded   15m
rook-ceph                     True    Release reconciliation succeeded   15m
rook-ceph-cluster             True    Release reconciliation succeeded   15m
object-bucket-claims          True    Release reconciliation succeeded   15m
```

Then, you can View Cluster Log Data (see page 940).

5.7.3.8 View Cluster Log Data

How to view the cluster's log data after enabling logging.

Though you enable logging at the Workspace level, viewing the log data is done at the cluster level, using the cluster's Grafana logging URL.

Run the following commands on the **management** cluster:

1. Execute the following command to get the namespace of your workspace:

   ```
   dkp get workspaces
   ```

   And copy the value under the `NAMESPACE` column for your workspace.

2. Set the `WORKSPACE_NAMESPACE` variable to the namespace copied in the previous step:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

   Run the following commands on the **attached** cluster to access the Grafana UI:

   Ensure you switched to the correct context or kubeconfig[622] of the attached cluster for the following kubectl commands:

3. Get the Grafana URL:

622 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

```
kubectl get ingress -n ${WORKSPACE_NAMESPACE} grafana-logging -o go-
template='https://{{with index .status.loadBalancer.ingress 0}}
{{or .hostname .ip}}{{end}}{{with index .spec.rules 0}}{{with index .http.paths
0}}{{.path }}{{end}}{{end}}{{"\n"}}'
```

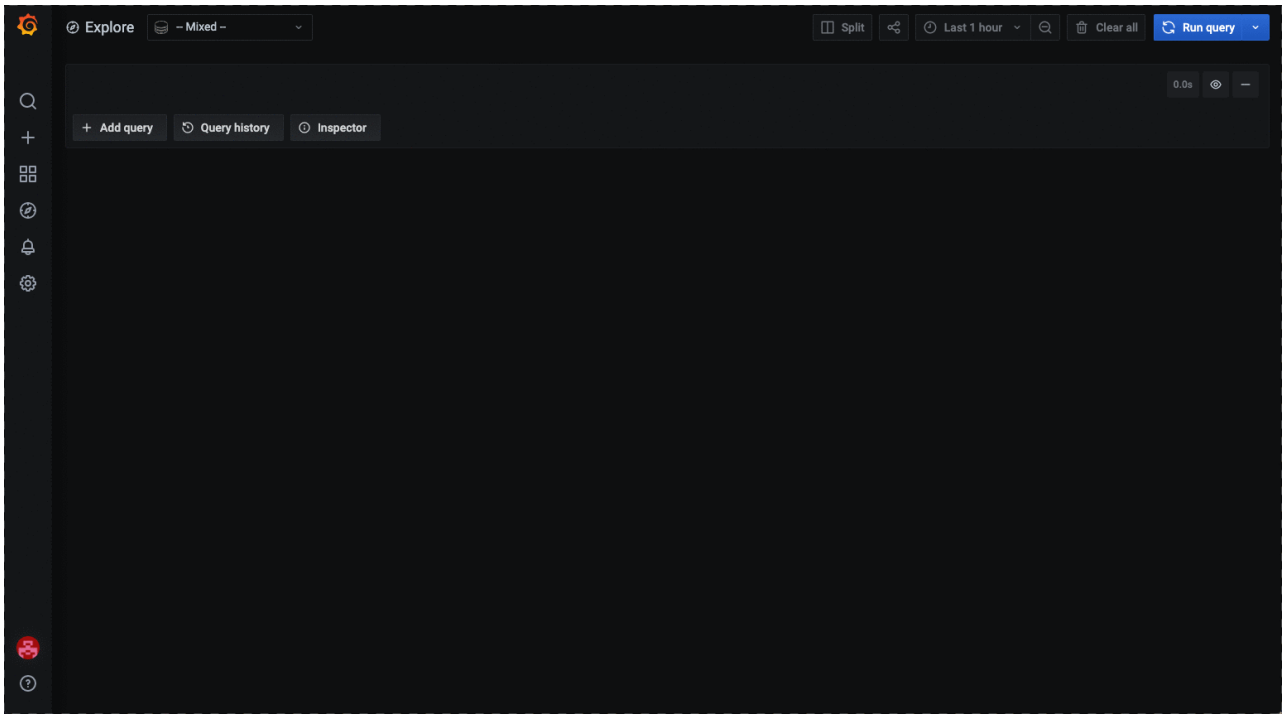
To view logs in Grafana:

1. Go to the Explore tab:

```
kubectl get ingress -n ${WORKSPACE_NAMESPACE} grafana-logging -o go-
template='https://{{with index .status.loadBalancer.ingress 0}}
{{or .hostname .ip}}{{end}}{{with index .spec.rules 0}}{{with index .http.paths
0}}{{.path }}{{end}}{{end}}/explore{{"\n"}}'
```

2. You may be prompted to log in using the SSO flow. See Authentication and Authorization (see page 964) for more information.
3. At the top of the page, change the datasource to `Loki`.

See the Grafana Loki documentation[623] for more on how to use the interface to view and query logs.



623 https://grafana.com/docs/grafana/v7.5/datasources/loki/

⚠ Cert-Manager and Traefik must be deployed in the attached cluster to be able to access the Grafana UI. These are deployed by default on the workspace.

5.7.4 Multi-Tenant Logging Overview

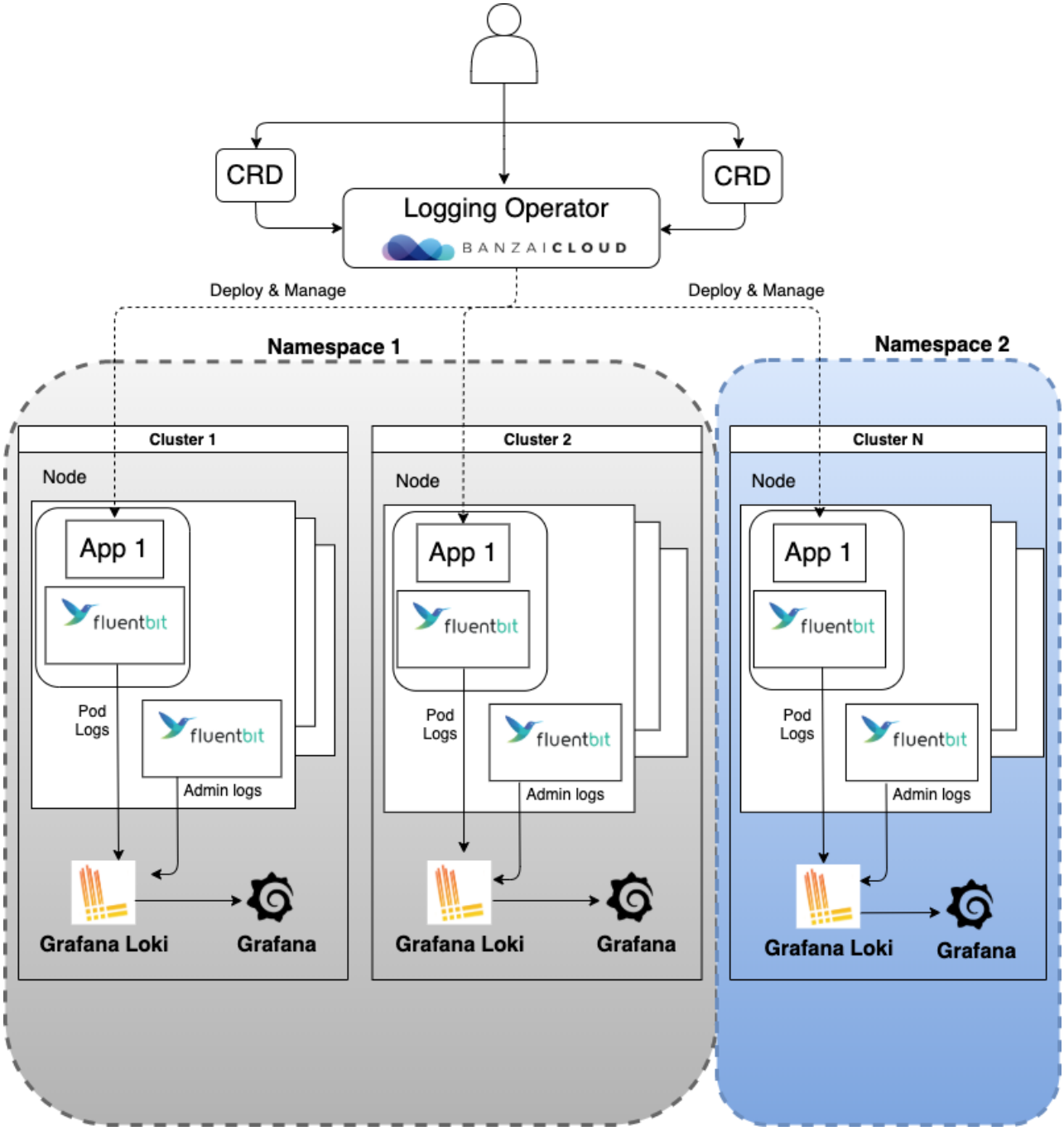
Enterprise Gov Advanced

Multi-tenant logging in Kommander is built on the DKP Logging architecture. Multi-tenant logging uses the same components as the base logging architecture:

- BanzaiCloud logging-operator
- Grafana Loki
- Grafana

⚠ You must perform the Workspace-level Logging (see page 931) procedures as a prerequisite to enable multi-tenant logging at the Project level as well.

Access to log data is done at the namespace level, through the use of Projects within Kommander as shown in the diagram:



Each Project namespace has a logging-operator “Flow” that sends its pod logs to its own Loki server. A custom controller deploys corresponding Loki and Grafana servers in each namespace, and defines a logging-operator Flow in each namespace that forwards its pod logs to its respective Loki server. There is a corresponding Grafana server for visualizations for each namespace.

For the convenience of cluster Administrators, a cluster-scoped Loki/Grafana instance pair is deployed with a corresponding Logging-operator `ClusterFlow` that directs pod logs from all namespaces to the pair. A cluster Administrator can grant access either to none of the logs, or to all logs collected from all pods in a

given namespace. Assigning teams to specific namespaces enables the team members to see only the logs for the namespaces they own.

As with any endpoint, if an Ingress controller is in use in the environment, take care that the ingress rules do not supersede the RBAC permissions and thus prevent access to the logs.

Cluster Administrators will need to monitor and adjust resource usage to prevent operational difficulties or excessive use on a *per namespace* basis.

5.7.4.1 Enable Multi-tenant Logging

Enterprise Gov Advanced

5.7.4.1.1 Prerequisites

Before you begin, you must:

- Enable Workspace-level Logging (see page 931) before you can configure multi-tenant logging.
- Be a cluster administrator with permissions to configure cluster level platform services.

5.7.4.1.2 Multi-tenant Logging Enablement Process

The steps required to enable multi-tenant logging include:

1. Create a Project. (see page 944)
2. Create the required Project-level AppDeployments. (see page 945)
3. Verify that the Project logging stack is installed. (see page 948)
4. View a Project's log data. (see page 949)

Get started with multi-tenant logging by Creating a Project (see page 944).

5.7.4.2 Create a Project for Logging

Enterprise Gov Advanced

To enable multi-tenant logging, you must first Create a Project (see page 716) and its namespace. Users assigned to this namespace will be able to access log data for only that namespace and not others.

Then, you can create project-level AppDeployments for use in multi-tenant logging (see page 945).

5.7.4.3 Create Project-level Logging AppDeployments

Enterprise Gov Advanced

How to create Project-level AppDeployments for use in multi-tenant logging

You must create AppDeployments in the Project namespace to enable and deploy the logging stack to all clusters within a Project. You can use the CLI to do this, or use the DKP UI to enable the logging applications.

To create the AppDeployments needed for Project-level logging, follow these steps **on the management cluster**:

1. Determine the name and namespace of the workspace that your project is in. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.

   ```
   dkp get workspaces
   ```

   Copy the values under the `NAME` and `NAMESPACE` columns for your workspace.

2. Export the `WORKSPACE_NAME` variable:

   ```
   export WORKSPACE_NAME=<WORKSPACE_NAME>
   ```

3. Export the `WORKSPACE_NAMESPACE` variable:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

4. Execute the following command to get the namespace of your project:

   ```
   kubectl get projects -n ${WORKSPACE_NAMESPACE}
   ```

   Copy the value under the `NAME` column for your project. This may NOT be identical to the Display Name of the `Project`.

5. Export the `PROJECT_NAME` variable:

   ```
   export PROJECT_NAME=<PROJECT_NAME>
   ```

6. Copy these commands and execute them from a command line:

```
dkp create appdeployment project-grafana-loki --app project-grafana-loki-0.69.1
6 --workspace ${WORKSPACE_NAME} --project ${PROJECT_NAME}
dkp create appdeployment project-grafana-logging --app project-grafana-
logging-6.57.4 --workspace ${WORKSPACE_NAME} --project ${PROJECT_NAME}
dkp create appdeployment project-logging --app project-logging-1.0.3 --
workspace ${WORKSPACE_NAME} --project ${PROJECT_NAME}
```

After completing these steps, you can Verify the Project Logging Stack Installation (see page 948)for multi-tenant logging.

5.7.4.4 Override ConfigMap to Modify the Storage Retention Period in Project Grafana Loki

Enterprise Gov Advanced

DKP configures Project Grafana Loki to retain log metadata and logs for 1 week, using the Compactor[624]. This retention policy can be customized.

> The minimum retention period is 24h.

To customize the retention policy using `configOverrides`, run these commands on the **management** cluster:

1. Determine the namespace of the workspace that your project is in. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.

   ```
   dkp get workspaces
   ```

   Copy the value under the `NAMESPACE` column for your workspace. This may NOT be identical to the Display Name of the `Workspace`.

2. Set the `WORKSPACE_NAMESPACE` variable to the namespace copied in the previous step:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

3. Get the namespace of your project:

   ```
   kubectl get projects --namespace ${WORKSPACE_NAMESPACE}
   ```

624 https://grafana.com/docs/loki/latest/operations/storage/boltdb-shipper/#compactor

Copy the value under the `PROJECT NAMESPACE` column for your workspace. This may NOT be identical to the Display Name of the `Project`.

4. Set the `PROJECT_NAMESPACE` variable to the namespace copied in the previous step:

```
export PROJECT_NAMESPACE=<PROJECT_NAMESPACE>
```

5. Create a `ConfigMap` with custom configuration values for Grafana Loki. Since the retention configuration is nested in a `config` string, you must copy the entire block. The following example sets the retention period under to 360 hours (15 days). Refer to Grafana Loki's Retention Configuration documentation[625] for more information about this field.

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: project-grafana-loki-custom-overrides
  namespace: ${PROJECT_NAMESPACE}
data:
  values.yaml: |
    loki:
      structuredConfig:
        limits_config:
          retention_period: 360h
EOF
```

6. Run the following command on the **management** cluster to reference the `configOverrides` in the `project-grafana-loki` AppDeployment:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: project-grafana-loki
  namespace: ${PROJECT_NAMESPACE}
spec:
  appRef:
    name: project-grafana-loki-0.48.6
    kind: ClusterApp
  configOverrides:
    name: project-grafana-loki-custom-overrides
EOF
```

625 https://grafana.com/docs/loki/latest/operations/storage/retention/#retention-configuration

5.7.4.5 Verify Project Logging Stack Installation

Enterprise Gov Advanced

How to verify the project logging stack installation for multi-tenant logging

You must wait for the Project's logging stack `HelmReleases` to deploy before configuring or using the Project-level logging features, including multi-tenancy:

Run the following commands on the **management** cluster:

1. Determine the namespace of the workspace that your project is in. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.

   ```
   dkp get workspaces
   ```

   Copy the value under the `NAMESPACE` column for your workspace.

2. Export the `WORKSPACE_NAMESPACE` variable:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

3. Execute the following command to get the namespace of your project

   ```
   kubectl get projects -n ${WORKSPACE_NAMESPACE}
   ```

   Copy the value under `PROJECT NAMESPACE` column for your project. This may NOT be identical to the Display Name of the `Project`.

4. Export the `PROJECT_NAMESPACE` variable:

   ```
   export PROJECT_NAMESPACE=<PROJECT_NAMESPACE>
   ```

   Run the following commands on the **managed** or **attached** cluster. For this, ensure you switch to the correct context or kubeconfig[626] of the cluster for the following `kubectl` commands:

5. Check the deployment status using this command on the attached cluster:

   ```
   kubectl get helmreleases -n ${PROJECT_NAMESPACE}
   ```

626 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

NOTE: It may take some time for these changes to take effect, based on the duration configured for the Flux GitRepository reconciliation.
When successfully deployed, you will see output that includes the following `HelmReleases`:

```
NAMESPACE              NAME                                 READY   STATUS
AGE
${PROJECT_NAMESPACE}   project-grafana-logging              True    Release
reconciliation succeeded   15m
${PROJECT_NAMESPACE}   project-grafana-loki                 True    Release
reconciliation succeeded   11m
${PROJECT_NAMESPACE}   project-loki-object-bucket-claims    True    Release
reconciliation succeeded   11m
```

Then, you can View Project Log Data (see page 949) within multi-tenant logging.

5.7.4.6 View Project Log Data

Enterprise Gov Advanced

How to view project log data within multi-tenant logging

You can only view the log data for a Project to which you have been granted access.

5.7.4.6.1 Access Project Grafana's UI

Run the following commands on the **management** cluster:

1. Determine the namespace of the workspace that your project is in. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.

   ```
   dkp get workspaces
   ```

   Copy the value under the `NAMESPACE` column for your workspace.

2. Export the `WORKSPACE_NAMESPACE` variable:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

3. Execute the following command to get the namespace of your project

   ```
   kubectl get projects -n ${WORKSPACE_NAMESPACE}
   ```

Copy the value under `PROJECT NAMESPACE` column for your project. This may NOT be identical to the Display Name of the `Project`.

4. Export the `PROJECT_NAMESPACE` variable:

```
export PROJECT_NAMESPACE=<PROJECT_NAMESPACE>
```

Run the following commands **on the attached cluster** to access the Project Grafana's UI:

Ensure you switched to the correct context or kubeconfig[627] of the attached cluster for the following kubectl commands:

5. Get the Grafana URL:

```
kubectl get ingress -n ${PROJECT_NAMESPACE} ${PROJECT_NAMESPACE}-project-
grafana-logging -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}{{with
index .spec.rules 0}}{{with index .http.paths 0}}{{.path }}{{end}}{{end}}
{{"\n"}}'
```

5.7.4.6.2 View Logs in Grafana

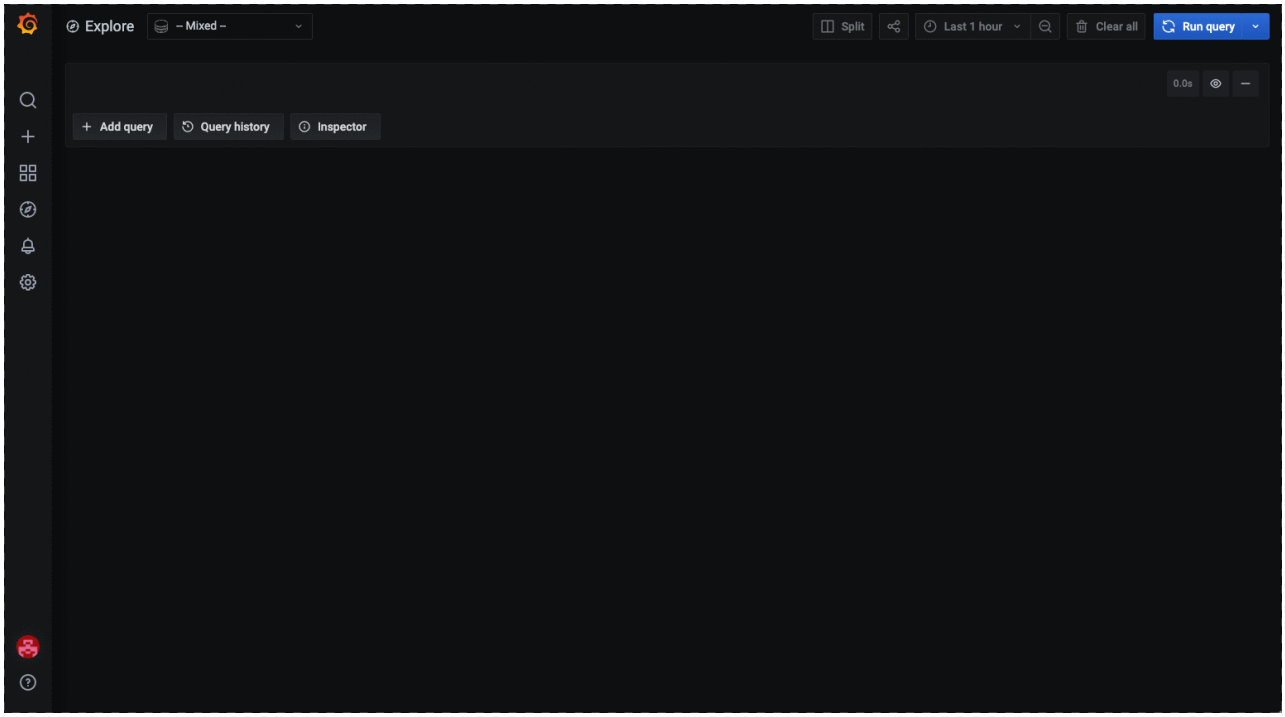
1. Go to the `Explore` tab:

```
kubectl get ingress -n ${PROJECT_NAMESPACE} ${PROJECT_NAMESPACE}-project-
grafana-logging -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}{{with
index .spec.rules 0}}{{with index .http.paths 0}}{{.path }}{{end}}{{end}}/
explore{{"\n"}}'
```

2. You may be prompted to log in using the SSO flow. See Authentication and Authorization (see page 964) for more information.
3. At the top of the page, change the data source to `Loki`.

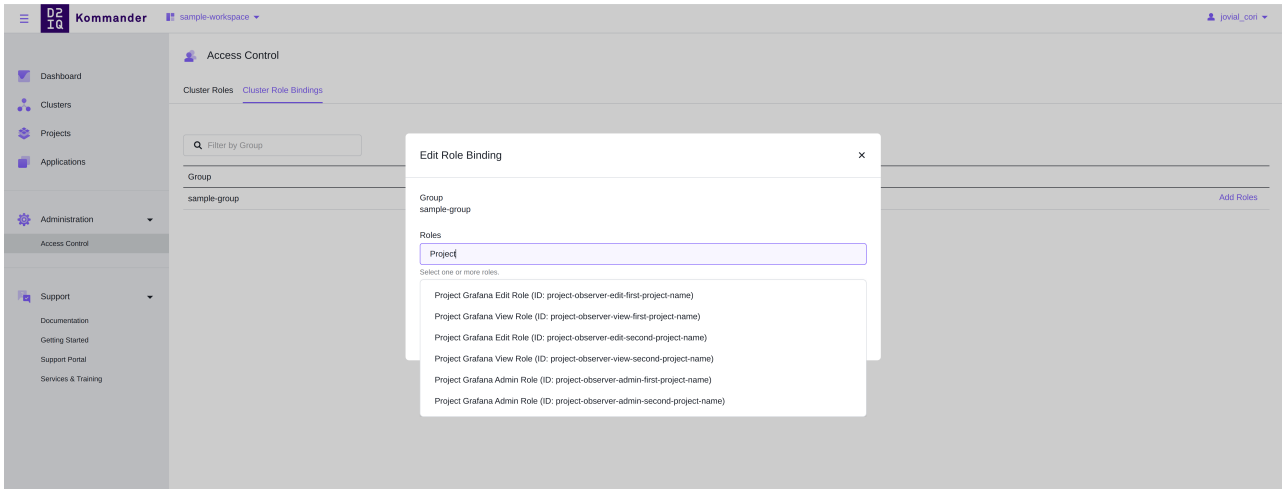
See the Grafana Loki documentation[628] for more on how to use the interface to view and query logs.

627 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/
628 https://grafana.com/docs/grafana/v7.5/datasources/loki/



> Cert-Manager and Traefik must be deployed in the attached cluster to be able to access the Grafana UI. These are deployed by default on the workspace.

You can configure workspace policy (see page 928) to restrict access to the Project logging Grafana UI. Each Grafana instance in a Project has a unique URL at the cluster level. Consider creating a `WorkspaceRoleBinding` that maps to a `ClusterRoleBinding`, on attached cluster(s), for each Project level Grafana instance. For example, If you have a group named `sample-group` and two projects named `first-project` and `second-project` in `sample-workspace` workspace, then the Role Bindings look similar to the following:



Select the correct role bindings for each group for a project at the workspace level.

5.7.5 Fluent Bit

Fluent Bit[629] is the DKP choice of open-source log collection and forwarding tool.

> ⚠ On the Management cluster, the Fluentbit application is disabled by default. The amount of admin logs ingested to Loki requires additional disk space to be configured on the `rook-ceph-cluster`. Enabling admin logs may use around 2GB/day per node. See Rook Ceph in DKP (see page 1034) for more details on how to configure the Rook Ceph Cluster.

5.7.5.1 Audit Log Collection

Auditing[630] in Kubernetes provides a way to chronologically document the actions taken on a cluster. On Kommander, by default, audit logs are collected and stored for quick indexing. Viewing and accessing can be done via the Grafana logging UI.

To adjust the default Audit Policy log backend[631] configuration, you must modify the log retention settings by Configuring the Control Plane (see page 1613) before creating the cluster. This needs to be done prior to creating the cluster since it cannot be edited after creation.

- Collecting systemd Logs from a Non-default Path (see page 952)

5.7.5.2 Related Information

For information on related topics or procedures, refer to the following:

- Logging (see page 928)

5.7.5.3 Collecting systemd Logs from a Non-default Path

By default, Fluent Bit pods are configured to collect `systemd` logs from the `/var/log/journal/` path on cluster nodes.

If `systemd-journald` running as a part of the OS on the nodes uses a different path for writing logs, you will need to override configuration of the `fluent-bit` AppDeployment to make Fluent Bit collect `systemd` logs.

629 https://fluentbit.io/
630 https://kubernetes.io/docs/tasks/debug-application-cluster/audit/
631 https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/#log-backend

To configure the Fluent Bit AppDeployment to collect `systemd` logs from a non-default path, follow these steps (all `kubectl` and `dkp` invocations refer to the **management** cluster):

1. Execute the following command to get the namespace of the workspace in which you would like to configure Fluent Bit:

   ```
   dkp get workspaces
   ```

   And copy the value under the `NAMESPACE` column for your workspace.

2. Set the `WORKSPACE_NAMESPACE` variable to the namespace copied in the previous step:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

3. Identify the `systemd-journald` log data storage path on the nodes of the clusters in the workspace by using the OS documentation and examining the `systemd` configuration.

   Usually it will be either `/var/log/journal` (typically used when `systemd-journald` is configured to store logs permanently; in this case the default Fluent Bit configuration should work) or `/run/log/journal` (typically used when `systemd-journald` is configured to use a volatile storage).

4. Extract the default Helm values used by the Fluent Bit App:

   ```
   kubectl get -n ${WORKSPACE_NAMESPACE} configmaps fluent-bit-0.20.9-d2iq-
   defaults -o=jsonpath='{.data.values\.yaml}' > fluent-bit-values.yaml
   ```

5. Edit the resulting file `fluent-bit-values.yaml` by removing all sections except for `extraVolumes`, `extraVolumeMounts` and `config.inputs`. The result should look similarly to this:

   ```
   extraVolumes:
   # we create this to have a persistent tail-db directory an all nodes
   # otherwise a restarted fluent-bit would rescrape all tails
   - name: tail-db
     hostPath:
       path: /var/log/tail-db
       type: DirectoryOrCreate
   # we create this to get rid of error messages that would appear on non control-
   plane nodes
   - name: kubernetes-audit
     hostPath:
       path: /var/log/kubernetes/audit
       type: DirectoryOrCreate
   ```

```
# needed for kmsg input plugin
- name: uptime
  hostPath:
    path: /proc/uptime
    type: File
- name: kmsg
  hostPath:
    path: /dev/kmsg
    type: CharDevice

extraVolumeMounts:
- name: tail-db
  mountPath: /tail-db
- name: kubernetes-audit
  mountPath: /var/log/kubernetes/audit
- name: uptime
  mountPath: /proc/uptime
- name: kmsg
  mountPath: /dev/kmsg

config:
  inputs: |
    # Collect audit logs, systemd logs, and kernel logs.
    # Pod logs are collected by the fluent-bit deployment managed by logging-
operator.
    [INPUT]
        Name tail
        Alias kubernetes_audit
        Path /var/log/kubernetes/audit/*.log
        Parser kubernetes-audit
        DB /tail-db/audit.db
        Tag audit.*
        Refresh_Interval 10
        Rotate_Wait 5
        Mem_Buf_Limit 135MB
        Buffer_Chunk_Size 5MB
        Buffer_Max_Size 20MB
        Skip_Long_Lines Off
    [INPUT]
        Name systemd
        Alias kubernetes_host
        DB /tail-db/journal.db
        Tag host.*
        Max_Entries 1000
        Read_From_Tail On
        Strip_Underscores On
    [INPUT]
        Name kmsg
        Alias kubernetes_host_kernel
        Tag kernel
```

6. Add the following item to the list under the `extraVolumes` key:

```
- name: kubernetes-host
  hostPath:
    path: <path to systemd logs on the node>
    type: Directory
```

7. Add the following item to the list under the `extraVolumeMounts` key:

```
- name: kubernetes-host
  mountPath: <path to systemd logs on the node>
```

These items will make Kubernetes mount systemd logs into Fluent Bit pods.

8. Add the following line into the `[INPUT]` entry identified by `Name systemd` and `Alias kubernetes_host`.

```
Path <path to systemd logs on the node>
```

This is needed to make Fluent Bit actually collect the mounted logs

9. Assuming that the path to systemd logs on the node is `/run/log/journal`, the result will look similarly to this:

```
extraVolumes:
# we create this to have a persistent tail-db directory an all nodes
# otherwise a restarted fluent-bit would rescrape all tails
- name: tail-db
  hostPath:
    path: /var/log/tail-db
    type: DirectoryOrCreate
# we create this to get rid of error messages that would appear on non control-
plane nodes
- name: kubernetes-audit
  hostPath:
    path: /var/log/kubernetes/audit
    type: DirectoryOrCreate
# needed for kmsg input plugin
- name: uptime
  hostPath:
    path: /proc/uptime
    type: File
- name: kmsg
  hostPath:
    path: /dev/kmsg
    type: CharDevice
- name: kubernetes-host
  hostPath:
```

```
    path: /run/log/journal
    type: Directory

extraVolumeMounts:
- name: tail-db
  mountPath: /tail-db
- name: kubernetes-audit
  mountPath: /var/log/kubernetes/audit
- name: uptime
  mountPath: /proc/uptime
- name: kmsg
  mountPath: /dev/kmsg
- name: kubernetes-host
  mountPath: /run/log/journal

config:
  inputs: |
    # Collect audit logs, systemd logs, and kernel logs.
    # Pod logs are collected by the fluent-bit deployment managed by logging-
operator.
    [INPUT]
        Name tail
        Alias kubernetes_audit
        Path /var/log/kubernetes/audit/*.log
        Parser kubernetes-audit
        DB /tail-db/audit.db
        Tag audit.*
        Refresh_Interval 10
        Rotate_Wait 5
        Mem_Buf_Limit 135MB
        Buffer_Chunk_Size 5MB
        Buffer_Max_Size 20MB
        Skip_Long_Lines Off
    [INPUT]
        Name systemd
        Alias kubernetes_host
        Path /run/log/journal
        DB /tail-db/journal.db
        Tag host.*
        Max_Entries 1000
        Read_From_Tail On
        Strip_Underscores On
    [INPUT]
        Name kmsg
        Alias kubernetes_host_kernel
        Tag kernel
```

10. Create a `ConfigMap` manifest with override values from `fluent-bit-values.yaml`:

```
cat <<EOF >fluent-bit-overrides.yaml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: fluent-bit-overrides
data:
  values.yaml: |
$(cat fluent-bit-values.yaml | sed 's/^/    /g')
EOF
```

11. Create a `ConfigMap` from the manifest above:

```
kubectl apply -f fluent-bit-overrides.yaml
```

12. Edit the `fluent-bit` AppDeployment to set the value of `spec.configOverrides.name` to the name of the created `ConfigMap`. (You can use the steps in the procedure, Deploy an Application with a Custom Configuration (see page 952) as a guide.)

```
dkp edit appdeployment -n ${WORKSPACE_NAMESPACE} fluent-bit
```

After your editing is complete, the AppDeployment resembles this example:

```
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: fluent-bit
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: fluent-bit-0.20.9
    kind: ClusterApp
  configOverrides:
    name: fluent-bit-overrides
```

13. Log in into the Grafana logging UI of your workspace and verify that logs with a label `log_source=kubernetes_host` are now present in Loki.

5.7.6 Scaling the Logging Stack

5.7.6.1 Introduction

Depending on the application workloads you run on your clusters, you may find that the default settings for the DKP logging stack do not meet your needs. In particular, if your workloads produce lots of log traffic, you

may find you need to adjust the logging stack components to properly capture all the log traffic. Follow the suggestions below to tune the logging stack components as needed.

5.7.6.2 Logging Operator

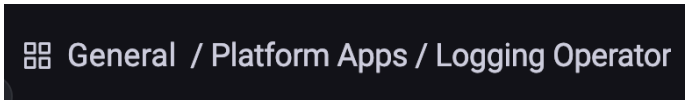
In a high log traffic environment, `fluentd` usually becomes the bottleneck of the logging stack. According to https://banzaicloud.com/docs/one-eye/logging-operator/operation/scaling/:

The typical sign of this is when `fluentd` *cannot handle its* ***buffer***[632] *directory size growth for more than the configured or calculated (timekey + timekey_wait) flush interval.*

For metrics to monitor, refer to https://docs.fluentd.org/monitoring-fluentd/monitoring-prometheus#metrics-to-monitor.

5.7.6.2.1 Grafana dashboard

In DKP, if the `Prometheus Monitoring` (kube-prometheus-stack) platform application is enabled, you can view the Logging Operator dashboard in the Grafana UI.



You can also improve fluentd throughput by disabling the buffering for `loki` clusterOutput.

5.7.6.2.2 Example Configuration

You can see an example configuration of the logging operator in Logging Stack Application Sizing Recommendations (see page 647).

For more information, refer to:

- https://docs.fluentd.org/deployment/performance-tuning-single-process

5.7.6.3 Grafana Loki

DKP deploys Loki in Microservice mode[633] – this provides you with the highest flexibility in terms of scaling.

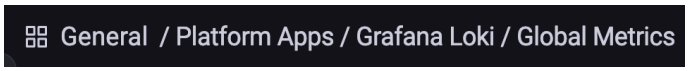
In a high log traffic environment, we recommend:

- Ingester should be the first component to be considered for scaling up.
- Distributor should be scaled up only when the existing Distributor is experiencing stress due to high computing resource usage.
    - Usually, the number of Distributor pods should be much lower than the number of Ingester pods

632 https://banzaicloud.com/docs/one-eye/logging-operator/configuration/plugins/outputs/buffer/
633 https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#microservices-mode

5.7.6.3.1 Grafana dashboard

In DKP, if `Prometheus Monitoring` (kube-prometheus-stack) platform app is enabled, you can view the Loki dashboards in Grafana UI and here is one of the Loki dashboard:



5.7.6.3.2 Example Configuration

You can see an example config of Loki at Logging Stack Application Sizing Recommendations (see page 647).

For more information, refer to:

- https://grafana.com/docs/loki/latest/fundamentals/architecture/components/
- https://grafana.com/docs/loki/latest/operations/scalability/
- https://grafana.com/docs/loki/latest/best-practices/

5.7.6.4 Rook Ceph

Ceph is the default S3 storage provider. In DKP, a Rook Ceph Operator and a Rook Ceph Cluster are deployed together to have a Ceph Cluster.

5.7.6.4.1 Storage

The default configuration of Rook Ceph Cluster in DKP has a 33% overhead in data storage for redundancy. Meaning, if the data disks allocated for your Rook Ceph Cluster is 1000Gb, 750Gb will be used to store your data. Thus, it is important to account for that in planning the capacity of your data disks to prevent issues.

5.7.6.4.1.1 ObjectBucketClaim storage limit

ObjectBucketClaim has a storage limit option to prevent your S3 bucket from growing over a limit. In DKP this is enabled by default.

Thus, after you size up your Rook Ceph Cluster for more storage, it is important to also increase the storage limit of your ObjectBucketClaims of your `grafana-loki` and/or `project-grafana-loki`.

To change it for `grafana-loki`, provide an override configmap in `rook-ceph-cluster` platform app to override `dkp.grafana-loki.maxSize`

To change it for `project-grafana-loki`, provide an override configmap in `project-grafana-loki` platform app to override `dkp.project-grafana-loki.maxSize`

5.7.6.4.2 Example Configuration

You can see an example config at Rook Ceph Cluster Sizing Recommendations (see page 652).

5.7.6.4.3 Ceph OSD CPU considerations

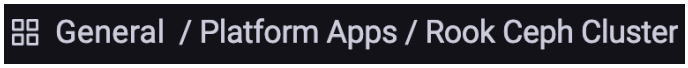
ceph-osd is the object storage daemon for the Ceph distributed file system. It is responsible for storing objects on a local file system and providing access to them over the network.

If you determine that the Ceph OSD component is the bottleneck, then you may wish to consider increasing the CPU allocated to it.

See this page for more info: https://ceph.io/en/news/blog/2022/ceph-osd-cpu-scaling/

5.7.6.4.4 Grafana dashboard

In DKP, if the `Prometheus Monitoring` (kube-prometheus-stack) platform app is enabled, you can view the Ceph dashboards in the Grafana UI. Below is one of the Ceph dashboard:



5.7.6.5 Audit Log

5.7.6.5.1 Overhead

Enabling audit logging requires additional computing and storage resources.
When you enable audit logging by enabling the `kommander-fluent-bit` AppDeployment, inbound traffic to the logging stack increases the log traffic by approximately 3-4 more times.

Thus, when enabling the audit log, consider scaling up all components in the logging stack mentioned above.

5.7.6.5.2 Fine-tuning audit log Fluent Bit

If you are certain that you only need to collect a subset of the logs that the default config makes the `kommander-fluent-bit` pods collect, you can add your own override configmap to `kommander-fluent-bit` with proper Fluent Bit `INPUT`, `FILTER`, `OUTPUT` settings. This helps reduce the audit log traffic.

To see the default configuration of Fluent Bit, see the Release Notes (see page 1946) > **Components and Applications**.

For more information, refer to:

- Admin-level logs (see page 930)
- https://docs.fluentbit.io/manual/administration/configuring-fluent-bit/classic-mode/configuration-file

5.7.7 Configuring Loki to use AWS S3 Storage in DKP

Follow the instructions on this page to configure Loki to use AWS S3 Storage in DKP

5.7.7.1 Configuring Loki

1. Execute the following command to get the namespace of your workspace:

```
dkp get workspaces
```

2. Set the `WORKSPACE_NAMESPACE` variable to the namespace copied in the previous step:

```
export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
```

3. Create a secret containing the static AWS S3 credentials. The secret is then mounted into each of the Grafana Loki pods as environment variables.

```
kubectl create secret generic dkp-aws-s3-creds -n${WORKSPACE_NAMESPACE} \
--from-literal=AWS_ACCESS_KEY_ID=<key id> \
--from-literal=AWS_SECRET_ACCESS_KEY=<secret key>
```

4. Create a config overrides ConfigMap to update the storage configuration:
NOTE: This can also be added to the installer configuration if you are configuring Grafana Loki on the Management Cluster

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: grafana-loki-overrides
  namespace: ${WORKSPACE_NAMESPACE}
data:
  values.yaml: |
    loki:
      annotations:
        secret.reloader.stakater.com/reload: dkp-aws-s3-creds
      structuredConfig:
        storage_config:
          aws:
            s3: s3://<region>/<bucket name>
    ingester:
      extraEnvFrom:
        - secretRef:
```

```
            name: dkp-aws-s3-creds
    querier:
      extraEnvFrom:
        - secretRef:
            name: dkp-aws-s3-creds
    queryFrontend:
      extraEnvFrom:
        - secretRef:
            name: dkp-aws-s3-creds
    compactor:
      extraEnvFrom:
        - secretRef:
            name: dkp-aws-s3-creds
    ruler:
      extraEnvFrom:
        - secretRef:
            name: dkp-aws-s3-creds
    distributor:
      extraEnvFrom:
        - secretRef:
            name: dkp-aws-s3-creds
EOF
```

5. Update the `grafana-loki` AppDeployment to apply the configuration override.

   NOTE: If you use the Kommander CLI installation configuration file, you don't need this step

```
cat << EOF | kubectl -n ${WORKSPACE_NAMESPACE} patch appdeployment grafana-loki
--type="merge" --patch-file=/dev/stdin
spec:
  configOverrides:
    name: grafana-loki-overrides
EOF
```

5.7.8 Customizing Logging Stack Applications

This page provides instructions on how you can customize the Logging Stack Applications in DKP.

5.7.8.1 Retrieve the Workspace Namespace

1. On the Management Cluster, execute the following command to get the namespace of your workspace:

```
dkp get workspaces
```

2. Copy the value under the `NAMESPACE` column for your workspace.

3. Set the `WORKSPACE_NAMESPACE` variable to the namespace copied in the previous step

5.7.8.1.1 How to customize Logging Stack Applications

1. On the Attached or Managed Cluster, retrieve the kubeconfig for the cluster.
2. Apply the ConfigMap directly to the managed/attached cluster using the name, `logging-operator-logging-overrides`:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: logging-operator-logging-overrides
  namespace: ${WORKSPACE_NAMESPACE}
data:
  values.yaml: |
    <insert config here>
EOF
```

5.7.8.1.2 Customized ConfigMap Example

This is an example of an ConfigMap that contains customized resource request and limit values for `fluentd`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: logging-operator-logging-overrides
  namespace: kommander
data:
  values.yaml: |
    fluentd:
      resources:
        limits:
          cpu: 1
          memory: 2000Mi
        requests:
          cpu: 1
          memory: 1500Mi
```

5.7.8.1.3 See Also

Scaling the Logging Stack (see page 957)

5.8 Security

Security architecture.

5.8.1 Authentication and authorization architecture

5.8.1.1 Details on distributed authentication and authorization between clusters

5.8.1.2 Authentication

DKP UI comes with a pre-configured authentication Dex[634] identity broker and provider.

> ⚠ Kubernetes, Kommander, and Dex do not store any user identities. The Kommander installation comes with default admin static credentials. These credentials should only be used to access the **DKP UI** for configuring an external identity provider. Currently, there is no way to update these credentials, so they should be treated as backup credentials and not used for normal access.

The DKP UI admin credentials are stored as a secret. They never leave the boundary of the UI cluster and are never shared to any other cluster.

The Dex service issues an OIDC ID token[635] on successful user authentication. Other platform services use the ID token as an authentication proof. User identity to the Kubernetes API server is provided by the `kube-oidc-proxy` platform service that reads the identity from an ID token. Web requests to DKP UI access are authenticated by the traefik forward auth[636] platform service.

634 https://github.com/dexidp/dex
635 https://openid.net/specs/openid-connect-core-1_0.html#IDToken
636 https://github.com/mesosphere/traefik-forward-auth

The kube-oidc-proxy[637] service authenticates kubectl CLI requests using the Kubernetes API Server Go library. This library requires that if an **email_verified** claim is present, it must be set to **true**, even if the `insecureSkipEmailVerified: true` flag is configured in the Dex connector. Thus, ensure that the OIDC provider is configured to set the email_verified field to 'true'.

A user identity is shared across a UI cluster and all other attached clusters.

5.8.1.2.1 Attached clusters

A newly attached cluster has federated `kube-oidc-proxy`, `dex-k8s-authenticator`, and `traefik-forward-auth` platform applications. These platform applications are configured to accept the Management/Essential (see page 79) cluster Dex issued ID tokens.

When the `traefik-forward-auth` is used as a Traefik Ingress authenticator[638], it checks if the user identity was issued by the Kommander cluster Dex service. An anonymous user is redirected to the Management/Essential cluster Dex service to authenticate and confirm their identity.

Never enter your own credentials on any of the attached clusters. On the the Management/Essential cluster use the static admin credentials or an external identity provider (IDP).

5.8.1.3 Authorization

There is no centralized authorization component in Kommander. Each component and service makes its own authorization decisions based on user identity.

5.8.2 OpenID Connect (OIDC) Introduction

5.8.2.1 An introduction to OpenID Connect (OIDC) Authentication in Kubernetes

All Kubernetes clusters have two categories of users: service accounts and normal users. Kubernetes manages authentication for service accounts, but the cluster administrator, or a separate service, manages authentication for normal users.

Kommander configures the cluster to use OpenID Connect (OIDC), a popular and extensible user authentication method, and installs Dex, a popular, open-source software product that integrates your existing Identity Providers with Kubernetes.

To begin, set up an Identity Provider with Dex, then use OIDC as the Authentication method.

637 https://github.com/jetstack/kube-oidc-proxy
638 https://docs.traefik.io/v2.4/providers/kubernetes-ingress/

5.8.2.2 Identity Provider

An Identity Provider (IdP) is a service that lets you manage identity information for users, including groups. A cluster created in Kommander uses Dex as its IdP. Dex, in turn, delegates to one or more external IdPs.

If you use already use one or more of the following IdPs, you can configure Dex to use them:

Name	Supports Refresh Tokens	Supports Groups Claim	Supports `preferred_username` Claim	Status	Notes
LDAP[639]	yes	yes	yes	stable	
GitHub[640]	yes	yes	yes	stable	
SAML 2.0[641]	no	yes	no	stable	
GitLab[642]	yes	yes	yes	beta	
OpenID Connect[643]	yes	yes	yes	beta	Includes Salesforce, Azure, etc.
Google[644]	yes	yes	yes	alpha	
LinkedIn[645]	yes	no	no	beta	
Microsoft[646]	yes	yes	no	beta	

639 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/ldap.md
640 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/github.md
641 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/saml.md
642 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/gitlab.md
643 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/oidc.md
644 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/google.md
645 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/linkedin.md
646 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/microsoft.md

Name	Supports Refresh Tokens	Supports Groups Claim	Supports `preferred_username` Claim	Status	Notes
AuthProxy[647]	no	no	no	alpha	Authentication proxies such as Apache2 mod_auth, etc.
Bitbucket Cloud[648]	yes	yes	no	alpha	
OpenShift[649]	no	yes	no	stable	

NOTE: These are the Identity Providers supported by Dex 2.22.0[650], the version used by DKP.

5.8.2.3 Add Login Connectors

Kommander uses Dex to provide OpenID Connect single sign-on (SSO) to the cluster. Dex can be configured to use multiple connectors, including GitHub, LDAP, and SAML 2.0. The Dex Connector documentation[651] describes how to configure different connectors. You can add the configuration as the values field in the Dex application. An example Dex configuration provided to the Kommander CLI's `install` command would look similar to this:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  dex:
    values: |
      config:
        connectors:
        - type: oidc
          id: google
          name: Google
          config:
            issuer: https://accounts.google.com/o/oauth2/v2/auth
            clientID: YOUR_CLIENT_ID
            clientSecret: YOUR_CLIENT_SECRET
```

647 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/authproxy.md
648 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/bitbucketcloud.md
649 https://github.com/dexidp/dex/blob/v2.22.0/Documentation/connectors/openshift.md
650 https://github.com/dexidp/dex/blob/v2.22.0/README.md
651 https://dexidp.io/docs/connectors/

```
            redirectURI: https://DKP_CLUSTER_DOMAIN/dex/callback
            scopes:
            - openid
            - profile
            - email
            insecureSkipEmailVerified: true
            insecureEnableGroups: true
            userIDKey: email
            userNameKey: email
[...]
```

5.8.2.4 Change the Access Token Lifetime

By default, the client access token lifetime is 24 hours. After this time, the token expires and cannot be used to authenticate. See the Dex documentation[652] for more information on access token expiration and rotation settings.

Here is an example configuration for extending the token lifetime to 48 hours:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  dex:
    values: |
      config:
        expiry:
          idTokens: "48h"
[...]
```

5.8.2.5 Authentication

OpenID Connect is an extension of the OAuth2 authentication protocol[653]. As required by OAuth2, the client must be registered with Dex. Do this by passing the name of the application and a callback/redirect URI. These handle the processing of the OpenID token after the user authenticates successfully. After registration, Dex returns a `client_id` and a `secret`. Authentication requests use these between the client and Dex to identify the client.

Users access Kommander in two ways:

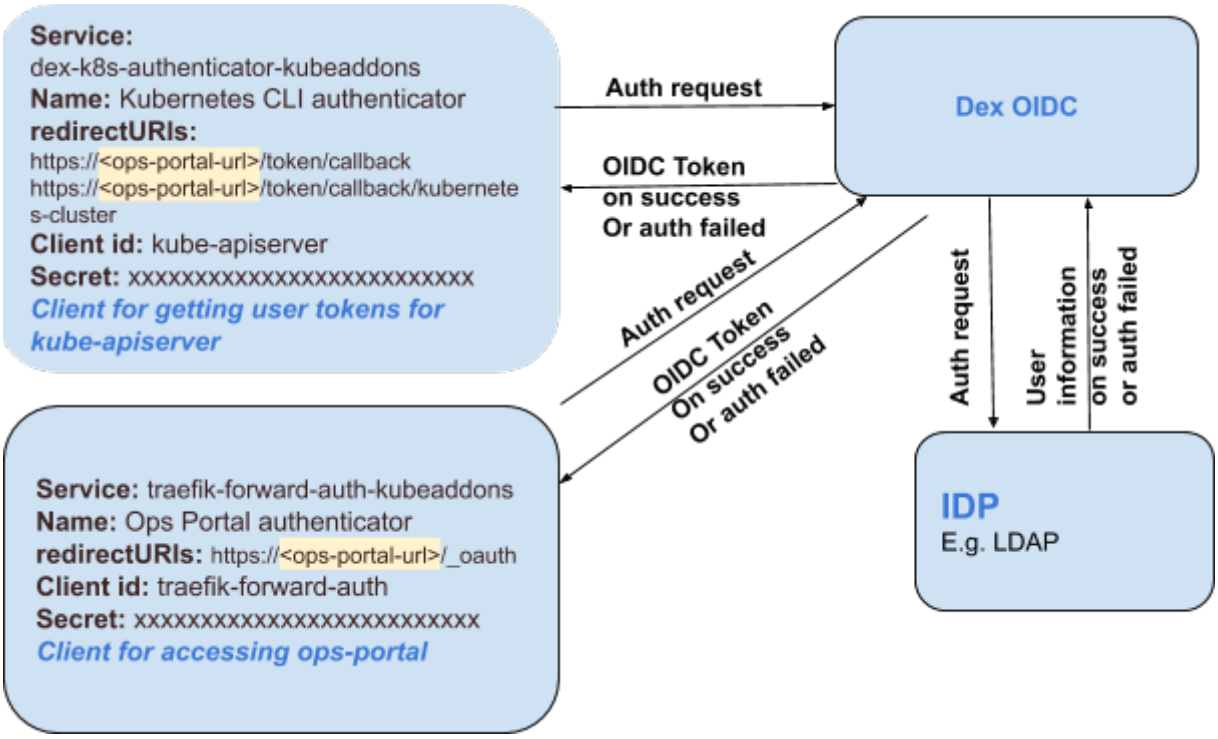
- To interact with Kubernetes API, usually through `kubectl`.
- To interact with the DKP UI, which has GUI dashboards for Prometheus, Grafana, etc.

In Kommander, Dex comes pre-configured with a client for these access use cases. The clients talk to Dex for authentication. Dex talks to the configured Identity Provider, or IdP, (for example LDAP, SAML, etc) to perform the actual task of authenticating the user.

652 https://dexidp.io/docs/tokens/#expiration-and-rotation-settings
653 https://oauth.net/2/

If the user authenticates successfully, Dex pulls the user's information from the IdP and forms an OpenID token. The token contains this information and returns it to the respective client's callback URL. The client or end user uses this token for communicating with the DKP UI or Kubernetes API respectively.

This figure illustrates these components and their interaction at a high level:



5.8.3 SAML Connector

5.8.3.1 Connect your Kommander cluster to an IdP using SAML

5.8.3.2 Connect Kommander to an IdP Using SAML

This procedure configures your Kommander cluster to use SAML, to connect to an identity provider (IdP).

1. Install DKP (see page 127).
2. Configure the IdP

   Provide the issuer URL and the Assertion Consumer Service (ACS) or callback URL to your IdP. The issuer URL points to the authentication endpoint at the service provider (Dex), which issues a request towards the IdP via the user agent.

   The issuer URL follows this schema:

   ```
   https://<your-cluster-host>/dex
   ```

The ACS URL points to the service provider (Dex) endpoint that receives SAML assertions issued by the IdP.

The ACS or callback URL should look like this:

```
https://<your-cluster-host>/dex/callback
```

Depending on the IdP, you might be asked to provide the configuration in some form of an XML snippet. See the following example, making sure to replace `<your-cluster-host>` with your URL:

```
<?xml version="1.0" encoding="UTF-8"?>
<EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata" entityID="https:
//<your-cluster-host>/dex">
    <SPSSODescriptor AuthnRequestsSigned="false" WantAssertionsSigned="true"
 protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
        <NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:persistent</
NameIDFormat>
        <AssertionConsumerService index="0" isDefault="true" Binding="urn:oasis
:names:tc:SAML:2.0:bindings:HTTP-POST" Location="https://<your-cluster-host>/
dex/callback" />
    </SPSSODescriptor>
</EntityDescriptor>
```

3. Modify the `dex` configuration:

   For this step, get the following from your IdP:

   - single sign-on URL or SAML URL -> `ssoURL`
   - base64 encoded, PEM encoded CA certificate -> `caData`
   - username attribute name in SAML response -> `usernameAttr`
   - email attribute name in SAML response -> `emailAttr`

   From above you need:

   - issuer URL -> `entityIssuer`
   - callback URL -> `redirectURI`

   Ensure you base64 encode the contents of the PEM file. As an example, the prefix of the contents will result into this exact base64 prefix:

   ```
   LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tC[...]
   ```

   You can add the configuration as the values field in the `dex` application. An example `dex` configuration provided to the Kommander CLI's install command (see page 1558) should look similar to:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  dex:
    values: |
      config:
      connectors:
      - type: saml
          id: saml
          name: SAML
          config:
            ssoURL: < url for POST request >
            caData: < base64 PEM encoded CA for the IdP server >
            redirectURI: https://<your-cluster-host>/dex/callback
            entityIssuer: https://<your-cluster-host>/dex
            usernameAttr: < user attribute in saml response >
            emailAttr: < email attribute in saml response >
[...]
```

4. Modify the `traefik-foward-auth-mgmt` configuration and add a whitelist:

   This step is required to give access to a user to the DKP UI. For each user, you must give Access to Kubernetes resources (see page 591) and add an entry in the `whitelist` below.

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
...
  traefik-forward-auth-mgmt:
    values: |
      traefikForwardAuth:
        allowedUser:
          valueFrom:
            secretKeyRef: null
        whitelist:
        - < allowed email addresses >
```

5. Run `kommander install --installer-config kommander.yaml` to deploy modified `dex`.
6. Visit `https://<your-cluster-host>/dkp/kommander/dashboard` to login to the DKP UI.
7. Select `Launch Console` and follow the authentication steps to complete the procedure.

5.8.4 Policy Controls

5.8.4.1 Workload Policy Controls

5.8.4.2 Enforce Policies Using Gatekeeper

5.8.4.2.1 Learn how to enforce policies using Gatekeeper

Gatekeeper[654] is the policy controller for Kubernetes, allowing organizations to enforce configurable policies using the Open Policy Agent[655], a policy engine for Cloud Native environments hosted by CNCF as a graduated-level project.

This tutorial describes how to use Gatekeeper to enforce policies by rejecting non-compliant resources. Specifically, this tutorial describes two constraints as a way to use Gatekeeper as an alternative to Pod Security Policies[656]:

- Prevent the running of privileged pods (see page 0)
- Prevent mounting host path volumes (see page 0)

5.8.4.2.2 Before you Begin

Before starting this tutorial, verify the following:

- You must have access to a Linux, macOS, or Windows computer with a supported operating system version.
- You must have a properly deployed and running cluster. For information about deploying Kubernetes with default settings on different types of infrastructures, see the Choose Infrastructure (see page 1050) topic.
- If you install Kommander with a custom configuration, make sure you enabled Gatekeeper.

If you intend to disable Gatekeeper, keep in mind that the app is deployed pre-configured with constraint templates that enforce multi-tenancy in projects.

654 https://github.com/open-policy-agent/gatekeeper
655 https://github.com/open-policy-agent/opa
656 https://kubernetes.io/docs/concepts/policy/pod-security-policy/

5.8.4.2.3 Use Gatekeeper

Gatekeeper uses the OPA Constraint Framework[657] to describe and enforce policy. Before you can define a constraint, you must first define a `ConstraintTemplate`, which describes both the `Rego` (a powerful query language) that enforces the constraint and the schema of the constraint. The schema of the constraint allows an admin to fine-tune the behavior of a constraint, much like arguments to a function.

The Gatekeeper repository includes a library of policies[658] to replace Pod Security Policies which you will use in the following tutorials.

5.8.4.2.3.1 Prevent privileged pods

Define the ConstraintTemplate

Create the privileged pod policy constraint template `k8spspprivilegedcontainer` by running the following command:

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper-library/master/library/pod-security-policy/privileged-containers/template.yaml
```

Define the Constraint

Constraints are then used to inform Gatekeeper that the admin wants to enforce a ConstraintTemplate, and how.

Create the privileged pod policy constraint `psp-privileged-container` by running the following command:

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper-library/master/library/pod-security-policy/privileged-containers/samples/psp-privileged-container/constraint.yaml
```

Test that the constraint is enforced

Create a privileged pod by running the following command:

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper-library/master/library/pod-security-policy/privileged-containers/samples/psp-privileged-container/example_disallowed.yaml
```

You should see the following output:

657 https://github.com/open-policy-agent/frameworks/tree/master/constraint
658 https://github.com/open-policy-agent/gatekeeper-library/tree/master/library/pod-security-policy

```
Error from server ([denied by psp-privileged-container] Privileged container is not
allowed: nginx, securityContext: {"privileged": true}): error when creating "https://
raw.githubusercontent.com/open-policy-agent/gatekeeper-library/master/library/pod-
security-policy/privileged-containers/samples/psp-privileged-container/
example_disallowed.yaml": admission webhook "validation.gatekeeper.sh" denied the
request: [denied by psp-privileged-container] Privileged container is not allowed:
nginx, securityContext: {"privileged": true}
```

5.8.4.2.3.2 Prevent host path volumes

Define the ConstraintTemplate

Create the host path volume policy constraint template `k8spsphostfilesystem` by running the following command:

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper-
library/master/library/pod-security-policy/host-filesystem/template.yaml
```

Define the Constraint

Constraints are then used to inform Gatekeeper that the admin wants to enforce a ConstraintTemplate, and how.

Create the host path volume policy constraint `psp-host-filesystem` by running the following command to only allow `/foo` to be mounted as a host path volume:

```
cat <<EOF | kubectl apply -f -
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sPSPHostFilesystem
metadata:
  name: psp-host-filesystem
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
  parameters:
    allowedHostPaths:
    - readOnly: true
      pathPrefix: "/foo"
EOF
```

Test that the constraint is enforced

Create a privileged pod by running the following command:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: nginx-host-filesystem
  labels:
    app: nginx-host-filesystem
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
      readOnly: true
  volumes:
  - name: cache-volume
    hostPath:
      path: /tmp # directory location on host
EOF
```

You should see the following output:

```
Error from server ([denied by psp-host-filesystem] HostPath volume {"hostPath": {"pat
h": "/tmp", "type": ""}, "name": "cache-volume"} is not allowed, pod: nginx-host-
filesystem. Allowed path: [{"readOnly": true, "pathPrefix": "/foo"}]): error when
creating "STDIN": admission webhook "validation.gatekeeper.sh" denied the request:
[denied by psp-host-filesystem] HostPath volume {"hostPath": {"path": "/tmp", "type":
""}, "name": "cache-volume"} is not allowed, pod: nginx-host-filesystem. Allowed
path: [{"readOnly": true, "pathPrefix": "/foo"}]
```

Test that the constraint allows the allowed host paths

Create a pod that mounts an allowed host path by running the following command:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: nginx-host-filesystem
  labels:
```

```
    app: nginx-host-filesystem
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
      readOnly: true
  volumes:
  - name: cache-volume
    hostPath:
      path: /foo # directory location on host
EOF
```

You should see the following output:

```
pod/nginx-host-filesystem created
```

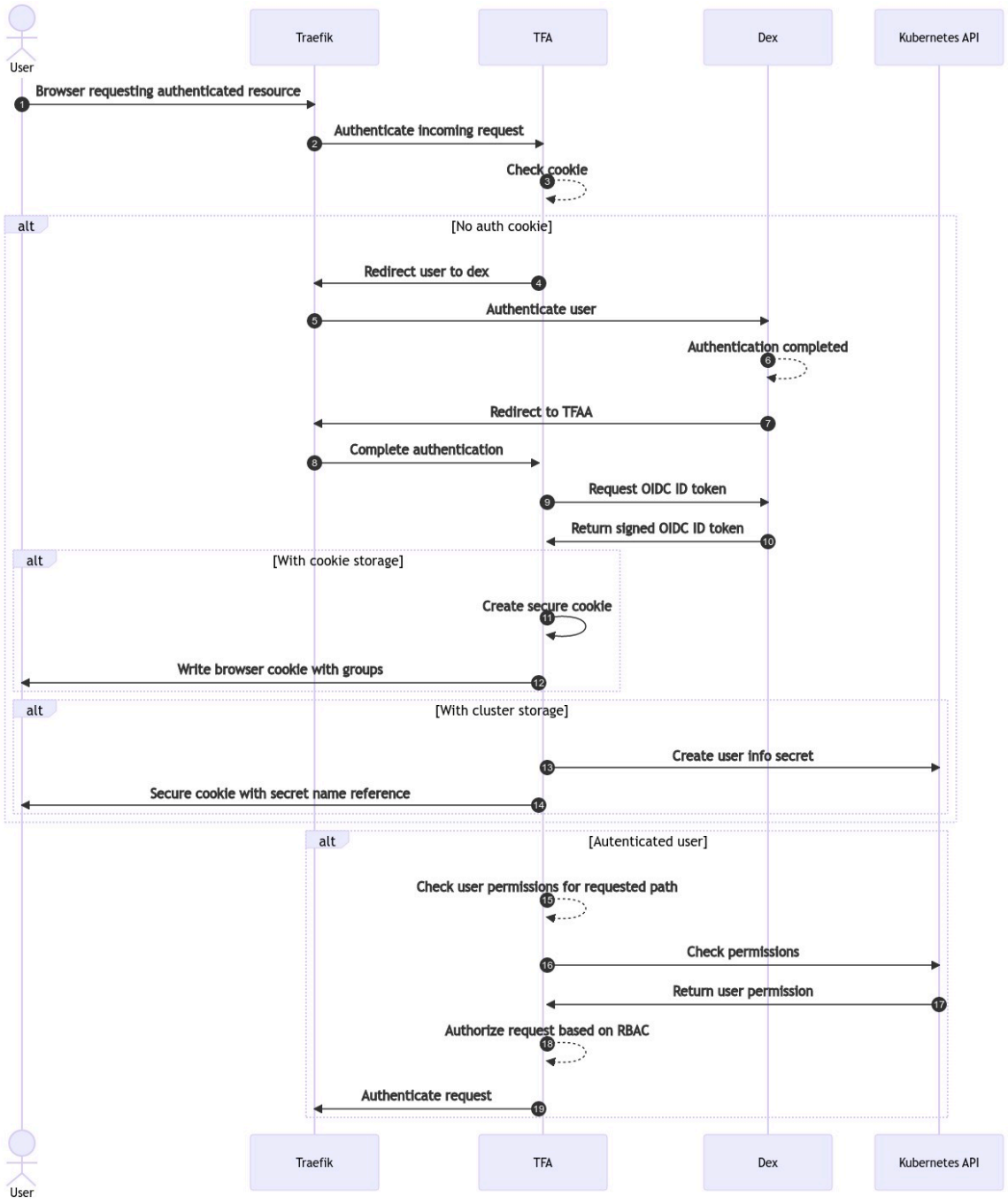
5.8.5 Traefik-Forward-Authentication in DKP (TFA)

5.8.5.1 TFA Overview

Traefik-Forward-Authentication (TFA) is a vital component in DKP. It handles authentication and authorization of web (HTTP) requests and how users can access the DKP UI. Regardless of the authentication method you choose (user and password, SSO, 2FA, etc.), TFA is a component that authorizes HTTP clients to enter your environment.

TFA is one of the standard applications in the Kommander component of DKP. It is deployed by a controller on all attached, managed, and management clusters.

5.8.5.2 TFA Authentication Workflow



5.8.5.3 Default TFA Configuration in DKP

In the default configuration in DKP, TFA stores all authentication information about users via the browser cookies. When TFA authenticates users, it stores the user's metadata in encrypted browser cookies.

Subsequent requests following initial authentication will use these cookies to recognize users without the need to re-authenticate them.

NOTE:

- The cookies are securely encrypted, so they cannot be modified by users.
- The cookies contain the RBAC username.
- The cookies contain a list of groups that users are associated with.

5.8.5.4 Cluster Storage Option

The browser cookie storage is limited to a maximum of 4Kb per cookie. However, if a user is assigned to a large number of groups, this limitation can be exceeded and this will return a response 500-internal server error, meaning that the user will be unable to access any web services.

To work around the cookie storage size limit, TFA can be configured to store the metadata claims in the cluster as a Kubernetes secret instead of in the browser. To do so, the `clusterStorage` option can be configured in the Traefik Forward Auth application when installing Kommander.

In order to enable the `clusterStorage` option, add the following to the cluster.yaml when installing Kommander:

```
traefik-forward-auth-mgmt:
  values: |
    clusterStorage:
      enabled: true
      namespace: kommander
```

If the `clusterStorage` feature is enabled, automatic garbage collection will delete the secrets after 12 hours. Keep in mind that enabling this feature will have performance implications for web requests, because TFA needs to load the secret to retrieve the user groups for each HTTP API request. Because of this, we recommend first trying to reduce the number of groups associated users and only enabling this option if that cannot be accomplished.

For additional information about traefik-forward-authentication, see the TFA page on Github[659].

5.8.6 Create Local Access

D2iQ recommends configuring an **external identity provider** to manage access to your cluster.

See Identity Providers (see page 609) for an overview of the benefits, supported providers, and instructions on how to configure an external identity provider.

659 https://github.com/mesosphere/traefik-forward-auth

However, you can create **local users** as an alternative, for example, when you want to quickly test RBAC policies. DKP automatically creates a unique local user, the `admin` user, but you can create additional local users and assign them other RBAC roles.

This section shows how to create and manage local users to access your Essential or Management cluster.

5.8.6.1 Next Step:

There are two ways in which you can create local users:

- Create Local Users during the Kommander Installation (see page 979)
- Create Local Users after Installing Kommander (see page 980)

5.8.6.2 Create Local Users during the Kommander Installation

Create local users before you install the Kommander component of DKP. If you have already installed Kommander, see Create Local Users after Installing Kommander (see page 980).

> ⚠ D2iQ does not recommend creating local users for production clusters. See Identity Providers (see page 609) for instructions on how to configure an external identity provider to manage your users.

1. Open the **Kommander Installer Configuration File** or `kommander.yaml` file. If you do not have the `kommander.yaml` file, initialize the configuration file (see page 1558), so you can edit it in the following steps.
   ⚠ Initialize this file only ONE time, otherwise you will overwrite previous customizations.
2. In that file, add the following customization for `dex` and create local users by establishing their credentials:
   ℹ Replace `<example_email>` with the user's email address or username.
   ℹ Replace `<password_bcrypt_hash>` with the bcrypt hash of the password you want to assign. You can use the htpasswd CLI to create the hash of a specific password. For example, by running `htpasswd -bnBC 10 "" password | tr -d ':\n' && echo`, you can generate the hash for the password “password”.

```
app:
  dex:
    enabled: true
    values: |
      config:
        staticPasswords:
```

```
    - email: <example_email>
      hash: <password_bcrypt_hash>
    - email: <example_email2>
      hash: <password_bcrypt_hash2>
```

3. Save the `kommander.yaml` file.
4. Install Kommander using the customized installation file. See https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29892161/Kommander+Customizations#install-config-file (see page 1560) for more information.
   After Kommander finishes installing, you have created local users, but you cannot use them until you have assigned them permissions (see page 982).

You have created a user that does not have any permissions to see or manage your DKP cluster yet. Add RBAC Roles to Local Users (see page 982) to complete the configuration.

5.8.6.2.1 Next Step:

Add RBAC Roles to Local Users (see page 982)

5.8.6.3 Create Local Users after Installing Kommander

Create local users after you install the Kommander component of DKP. If you have not installed Kommander yet, and want to create additional users during the installation, see Create Local Users during the Kommander Installation (see page 979).

D2iQ does not recommend creating local users for production clusters. See Identity Providers (see page 609) for instructions on how to configure an external identity provider to manage your users.

Customize the Dex AppDeployment, and add a `configOverrides` section.

1. Create a `configMap` resource with the credentials of the new local user:
   Replace `<example_email>` with the user's email address or a username.
   Replace `<password_bcrypt_hash>` with the bcrypt hash of the password you want to assign. You can use the htpasswd CLI to create the hash of a specific password. For example, by running `htpasswd -bnBC 10 "" password | tr -d ':\n' && echo` you can generate the hash for the password “password”.

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: dex-overrides
  namespace: kommander
data:
  values.yaml: |
    config:
      staticPasswords:
      - email: <example_email>
        hash: <password_bcrypt_hash>
EOF
```

2. Open the Dex AppDeployment to edit it:

```
kubectl edit -n kommander appdeployment dex
```

The editor displays the AppDeployment.

3. Copy the following values and paste them in a location in the file where they are nested in the `spec` field:

```
  configOverrides:
    name: dex-overrides
```

Example:

```
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
...
spec:
  appRef:
    kind: ClusterApp
    name: dex-2.11.1
  clusterConfigOverrides:
  - clusterSelector:
      matchExpressions:
      - key: kommander.d2iq.io/cluster-name
        operator: In
        values:
        - host-cluster
    configMapName: dex-kommander-overrides
  configOverrides: # Copy and paste this section.
    name: dex-overrides
status:
```

```
...
```

Editing the AppDeployment restarts the HelmRelease for Dex. The new users will be created after the reconciliation. However, the user creation is not completed until you assign it permissions (see page 982).

You have created a user that does not have any permissions to see or manage your DKP cluster yet. Add RBAC Roles to Local Users (see page 982) to complete the configuration.

5.8.6.3.1 Next Step:

Add RBAC Roles to Local Users (see page 982)

5.8.6.4 Add RBAC Roles to Local Users

Manage access to your cluster and its resources by assigning Kubernetes RBAC roles to local users. If you have not created local users yet, see Create Local Users after Installing Kommander (see page 980) or Create Local Users during the Kommander Installation (see page 979).

5.8.6.4.1 Assign a Role - Cluster Admin Example

Create the following `ClusterRoleBinding` resource:

- Replace `<example_email>` with the user's email address or a username.
- Replace `cluster-admin` with the RBAC role you want to assign to a user.
- If you have configured an Identity Provider for a specific workspace (see page 714), configure the `subjects.name` field to `<workspace_ID>:<user_email>`. For example, `tenant-z:jane.doe@example.com`.

```
cat <<EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
```

```
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: <example_email>
EOF
```

After assigning the previous role to `<example_email>`, the user is able to log in to the cluster using the credentials you assigned in Create Local Users after Installing Kommander (see page 980) or Create Local Users during the Kommander Installation (see page 979).

The **Login page** and cluster **URL** is the same for the default admin user and the local users you create with this method.

5.8.6.4.2 More Information

For more information on RBAC resources in DKP, see Granting Access to Kubernetes and Kommander Resources (see page 596).

For general information on RBAC as a Kubernetes resource, see the official Kubernetes RBAC[660] documentation.

5.8.6.4.3 Next Topic:

Modify or Delete Local Users (see page 983)

5.8.6.5 Modify or Delete Local Users

5.8.6.5.1 Modify Local Users

To change the password or username of a user, edit the `dex-overrides` ConfigMap.

If you change the email address or username of a user, ensure you update any `RoleBindings` or RBAC resources associated with this user.

5.8.6.5.2 Delete Local Users

To delete local users, edit the `dex-overrides` ConfigMap and remove the email and hash fields for the user. Also, ensure you delete any `RoleBindings` or RBAC resources associated with this user.

660 https://kubernetes.io/docs/reference/access-authn-authz/rbac

5.9 Networking

5.9.1 Configure networking for Konvoy cluster

This section describes different networking components that come together to form a Konvoy networking stack. It assumes familiarity with Kubernetes networking.

5.9.2 Networking Service

A Service[661] is an API resource that defines a logical set of pods and a policy by which to access them, and is an abstracted manner to expose applications as network services.

Kubernetes gives pods their own IP addresses and a single DNS name for a set of pods. Services are used as entrypoints to load-balance the traffic across the pods. A selector determines the set of Pods targeted by a Service.

For example, if you have a set of pods that each listen on TCP port `9191` and carry a label `app=MyKonvoyApp`, as configured in the following:

```
apiVersion: v1
kind: Service
metadata:
  name: my-konvoy-service
  namespace: default
spec:
  selector:
    app: MyKonvoyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9191
```

This specification creates a new `Service` object named `"my-konvoy-service"`, that targets TCP port `9191` on any pod with the `app=MyKonvoyApp` label.

661 https://kubernetes.io/docs/concepts/services-networking/service/#service-resource

Kubernetes assigns this Service an IP address. In particular, the `kube-proxy` implements a form of virtual IP for Services of type other than `ExternalName`.

- The name of a Service object must be a valid DNS label name.
- A Service is not a Platform Service.

5.9.2.1 Service Topology

Service Topology[662] is a mechanism in Kubernetes to route traffic based upon the Node topology of the cluster. For example, you can configure a Service to route the traffic to endpoints on specific nodes, or even based on the region or availability zone of the node's location.

To enable this new feature in your Kubernetes cluster, use the feature gates `--feature-gates="ServiceTopology=true,EndpointSlice=true"` flag. After enabling, you can control Service traffic routing by defining the `topologyKeys` field in the Service API object.

In the following example, a Service defines `topologyKeys` to be routed to endpoints only in the same zone:

```
apiVersion: v1
kind: Service
metadata:
  name: my-konvoy-service
  namespace: default
spec:
  selector:
    app: MyKonvoyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9191
  topologyKeys:
  - "topology.kubernetes.io/zone"
```

If the value of the `topologyKeys` field does not match any pattern, the traffic is rejected.

662 https://kubernetes.io/docs/concepts/services-networking/service-topology/#using-service-topology

5.9.2.2 EndpointSlices

EndpointSlices[663] are an API resource that appears as a scalable and more manageable solution to network endpoints within a Kubernetes cluster. They allow for distributing network endpoints across multiple resources with a limit of 100 endpoints per EndpointSlice.

An EndpointSlice contains references to a set of endpoints, and the control plane takes care of creating EndpointSlices for any Service that has a selector specified. These EndpointSlices include references to all the pods that match the Service selector.

Like Services, the name of a EndpointSlice object must be a valid DNS subdomain name.

In this example, here's a sample EndpointSlice resource for the example Kubernetes Service:

```
apiVersion: discovery.k8s.io/v1beta1
kind: EndpointSlice
metadata:
  name: konvoy-endpoint-slice
  namespace: default
  labels:
    kubernetes.io/service-name: my-konvoy-service
addressType: IPv4
ports:
- name: http
  protocol: TCP
  port: 80
endpoints:
- addresses:
  - "192.168.126.168"
  conditions:
    ready: true
  hostname: ip-10-0-135-39.us-west-2.compute.internal
  topology:
    kubernetes.io/hostname: ip-10-0-135-39.us-west-2.compute.internal
    topology.kubernetes.io/zone: us-west2-b
```

5.9.2.3 DNS for Services and Pods

Every new Service object in Kubernetes gets assigned a DNS name. The Kubernetes DNS component schedules a DNS name for the pods and services created on the cluster, and then the Kubelets are configured so containers can resolve these DNS names.

Considering previous examples, assume there is a Service named `my-konvoy-service` in the Kubernetes namespace `default`. A Pod running in namespace `default` can look up this service by performing a DNS query for `my-konvoy-service`. A Pod running in namespace `kommander` can look up this service by performing a DNS query for `my-konvoy-service.default`.

663 https://kubernetes.io/docs/concepts/services-networking/endpoint-slices/#endpointslice-resource

In general, a pod has the following DNS resolution:

```
pod-ip-address.namespace-name.pod-name.cluster-domain.example.
```

Similarly, a service has the following DNS resolution:

```
service-name.namespace-name.svc.cluster-domain.example.
```

You can find additional information about all the possible record types and layout here[664].

5.9.2.4 Ingress and Networking

Ingress[665] is an API resource that manages external access to the services in a cluster through HTTP or HTTPS. It offers name-based virtual hosting, SSL termination and load balancing when exposing HTTP/HTTPS routes from outside to services in the cluster.

The traffic policies are controlled by rules as part of the Ingress definition. Each rule defines the following details:

- An optional host to which apply the rules.
- A list of paths or routes which has an associated backend defined with a Service `name`, a port `name` and `number`.
- A backend is a combo of a Service and port names, or a custom resource backend defined as a CRD. Consequently HTTP/HTTPS requests to the Ingress that matches the host and path of the rule are sent to the listed backend.

An example of an Ingress specification is:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: konvoy-ingress
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /path
        pathType: Prefix
        backend:
          service:
            name: my-konvoy-service
            port:
```

664 https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/
665 https://kubernetes.io/docs/concepts/services-networking/ingress/#what-is-ingress

```
            number: 80
```

In Kommander, you can expose services to the outside world using Ingress objects.

5.9.2.4.1 Ingress Controllers

In contrast with the controllers in the Kubernetes control plane, Ingress controllers are not started with a cluster so you need to choose the desired Ingress controller.

An Ingress controller has to be deployed in a cluster for the Ingress definitions to work.

Kubernetes as a project currently supports and maintains GCE and nginx controllers.

These are four of the most known Ingress controllers[666]:

- HAProxy Ingress[667] is a highly customizable community-driven ingress controller for HAProxy.
- NGINX offers support and maintenance for the NGINX Ingress Controller[668] for Kubernetes.
- Traefik[669] is a fully featured Ingress controller (Let's Encrypt, secrets, http2, websocket), and has commercial support.
- Ambassador API Gateway[670] EXPERIMENTAL is an Envoy based Ingress controller with community and commercial support.

In Kommander, `Traefik` deploys by default as a well-suited Ingress controller.

5.9.2.5 Network Policies

NetworkPolicy is an API resource that controls the traffic flow at port level 3 or 4, or at the IP address level. It enables defining constraints on how a pod communicates with various network services such as `endpoints` and `services`.

A Pod can be restricted to talk to other network services through a selection of the following identifiers:

- Namespaces that have to access. There can be pods that are not allowed to talk to other namespaces.
- Other allowed IP blocks regardless of the node or IP address assigned to the targeted Pod.
- Other allowed Pods.

An example of a NetworkPolicy specification is:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: network-konvoy-policy
  namespace: default
```

666 https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/#additional-controllers
667 https://haproxy-ingress.github.io/
668 https://www.nginx.com/products/nginx-ingress-controller
669 https://github.com/containous/traefik
670 https://www.getambassador.io/

```
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          app: MyKonvoyApp
    - podSelector:
        matchLabels:
          app: MyKonvoyApp
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

As shown in the example, when defining a pod or namespace based NetworkPolicy, you use a selector to specify what traffic is allowed to and from the Pod(s).

5.9.2.5.1 Adding Entries to Pod /etc/hosts with HostAliases

The Pod API resource definition has a `HostAliases` field that allows adding entries to the Pod's container `/etc/hosts` file. This field overrides the hostname resolution when DNS and other options are not applicable.

For example, to resolve `foo.node.local`, `bar.node.local` to `127.0.0.1` and `foo.node.remote`, `bar.node.remote` to `10.1.2.3`, configure the `HostAliases` values as follows:

```
apiVersion: v1
kind: Pod
metadata:
  name: hostaliases-konvoy-pod
spec:
  restartPolicy: Never
  hostAliases:
```

```
  - ip: "127.0.0.1"
    hostnames:
    - "foo.node.local"
    - "bar.node.local"
  - ip: "10.1.2.3"
    hostnames:
    - "foo.node.remote"
    - "bar.node.remote"
  containers:
  - name: cat-hosts
    image: busybox
    command:
    - cat
    args:
    - "/etc/hosts"
```

5.9.3 Required Domains

5.9.3.1 This section describes the required domains for DKP

You must have access to the following domains through the customer networking rules so that Kommander can download all required images:

- http://docker.io
- http://gcr.io
- http://k8s.gcr.io
- mcr.microsoft.com[671]
- nvcr.io
- http://quay.io
- http://us.gcr.io
- registry.k8s.io[672]

In an air-gapped installation, these domains do not need to be accessible.

671 http://mcr.microsoft.com
672 http://registry.k8s.io

5.9.4 Configure Ingress for load balancing

5.9.4.1 Learn how to configure Ingress settings for load balancing (layer-7)

Ingress is the name used to describe an API object that manages external access to the services in a cluster. Typically, an Ingress exposes HTTP and HTTPS routes from outside the cluster to services running within the cluster.

The object is called an Ingress because it acts as a gateway for inbound traffic. The Ingress receives inbound requests and routes them according to the rules you defined for the **Ingress resource** as part of your cluster configuration.

Expose an application running on your cluster by configuring an Ingress for load balancing (layer-7).

5.9.4.2 Prerequisites

Before you begin, you must:

- Have access to a Linux, macOS, or Windows computer with a supported operating system version.
- Have a properly deployed and running cluster.

5.9.4.3 Expose a pod using an Ingress (L7)

1. Deploy two web application Pods on your Kubernetes cluster by running the following command:

```
kubectl run --restart=Never --image hashicorp/http-echo --labels app=http-
echo-1 --port 80 http-echo-1 -- -listen=:80 --text="Hello from http-echo-1"
kubectl run --restart=Never --image hashicorp/http-echo --labels app=http-
echo-2 --port 80 http-echo-2 -- -listen=:80 --text="Hello from http-echo-2"
```

2. Expose the Pods with a service type of ClusterIP by running the following commands:

```
kubectl expose pod http-echo-1 --port 80 --target-port 80 --name "http-echo-1"
kubectl expose pod http-echo-2 --port 80 --target-port 80 --name "http-echo-2"
```

3. Create the Ingress to expose the application to the outside world by running the following command:

```
cat <<EOF | kubectl create -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: kommander-traefik
    traefik.ingress.kubernetes.io/router.tls: "true"
```

```
  generation: 7
  name: echo
  namespace: default
spec:
  rules:
  - http:
      paths:
      - backend:
          service:
            name: http-echo-1
            port:
              number: 80
        path: /echo1
        pathType: Prefix
  - http:
      paths:
      - backend:
          service:
            name: http-echo-2
            port:
              number: 80
        path: /echo2
        pathType: Prefix
EOF
```

The configuration settings in this example illustrate:

- setting the `kind` to `Ingress`.
- setting the `service.name` to be exposed as each `backend`.

4. Run the following command to get the URL of the load balancer created on AWS for the Traefik service:

```
kubectl get svc kommander-traefik -n kommander
```

This command displays the internal and external IP addresses for the exposed service. (Note that IP addresses and host names are for illustrative purposes. Always use the information from your own cluster)

```
NAME                TYPE           CLUSTER-IP    EXTERNAL-IP
PORT(S)                                  AGE
kommander-traefik   LoadBalancer   10.0.24.215
abf2e5bda6ca811e982140acb7ee21b7-37522315.us-west-2.elb.amazonaws.com   80:3116
9/TCP,443:32297/TCP,8080:31923/TCP   4h22m
```

5. Validate that you can access the web application Pods by running the following commands: (Note that IP addresses and host names are for illustrative purposes. Always use the information from your own cluster)

```
curl -k https://abf2e5bda6ca811e982140acb7ee21b7-37522315.us-
west-2.elb.amazonaws.com/echo1
curl -k https://abf2e5bda6ca811e982140acb7ee21b7-37522315.us-
west-2.elb.amazonaws.com/echo2
```

5.9.5 Ingress

5.9.5.1 Traefik Ingress Controller

Kubernetes Ingress resources expose HTTP and HTTPS routes from outside the cluster to services within the cluster. In Kommander, the Traefik[673] Ingress controller is installed by default and provides access to the DKP UI.

An Ingress performs the following:

- Gives Services externally-reachable URLs
- Load balances traffic
- Terminates SSL/TLS sessions
- Offers name-based virtual hosting

An Ingress controller fulfills the Ingress with a load balancer.

A cluster can have multiple Ingress controllers. D2iQ recommends adding your own Ingress controllers for your applications. The Traefik Ingress controller that Kommander installs for access to the DKP UI can be replaced later if a different solution is a better fit. Using your own Ingress controller in parallel for your own business requirements ensures that you are not limited by any future changes in Kommander.

5.9.5.2 Traefik v2.4

Traefik is a modern HTTP reverse proxy and load balancer that deploys microservices with ease. Kommander currently installs Traefik v2.4 by default on every cluster. Traefik creates a service of type `LoadBalancer`. In the cloud, the cloud provider creates the appropriate load balancer. In an on-premises deployment, by default, it uses MetalLB.

Traefik listens to the Kubernetes API and automatically generates and updates the routes without any further configuration or intervention so that the Services selected by the Ingress resources are connected to the outside world. Further, Traefik supports a rich set of functionality such as Name-based routing, Path-based routing, Traffic splitting, etc.

Major features highlighted in the Traefik documentation:

- Continuously updates its configuration (No restarts!)
- Supports multiple load balancing algorithms
- Provides HTTPS to your microservices
- Circuit breakers, retry
- A clean web UI
- Websocket, HTTP/2, GRPC ready

673 https://landscape.cncf.io/card-mode?category=service-proxy&grouping=category&selected=traefik

- Provides metrics (Rest, Prometheus, Datadog, StatsD, InfluxDB)
- Keeps access logs (JSON, CLF)
- Exposes a Rest API
- Packaged as a single binary file (made with go) and available as a docker image

5.9.5.3 Related Information

For information on related topics or procedures, refer to the following:

- List of Ingress controllers[674]
- Traefik v2.4 docs[675]
- Configure Ingress for load balancing (see page 991)
- Load Balancing (see page 994)

5.9.6 Load Balancing

In a Kubernetes cluster, depending on the flow of traffic direction, there are two kinds of load balancing:

- Internal load balancing for the traffic within a Kubernetes cluster
- External load balancing for the traffic coming from outside the cluster

5.9.6.1 Load Balancing for Internal Traffic

Load balancing within a Kubernetes cluster is accessed through a `ClusterIP` service type. `ClusterIP` presents a single IP address to the client and load balances the traffic going to this IP to the backend servers. The actual load balancing happens using `iptables` or IPVS configuration. The `kube-proxy` Kubernetes component programs these. The `iptables` mode of operation uses DNAT[676] rules to distribute direct traffic to real servers, whereas IPVS[677] leverages in-kernel transport-layer load-balancing. Read a comparison between these two methods[678]. By default, `kube-proxy` runs in `iptables` mode. The kube-proxy configuration can be altered by updating the `kube-proxy` configmap in the `kube-system` namespace.

5.9.6.2 Load Balancing for External Traffic

External traffic destined for the Kubernetes service requires a service of type `LoadBalancer`, through which external clients connect to your internal service. Under the hood, it uses a load balancer provided by the underlying infrastructure to direct the traffic.

674 https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/
675 https://doc.traefik.io/traefik/v2.4/
676 https://www.linuxtopia.org/Linux_Firewall_iptables/x4013.html
677 https://en.wikipedia.org/wiki/IP_Virtual_Server
678 https://www.projectcalico.org/comparing-kube-proxy-modes-iptables-or-ipvs/

In DKP environments, the external load balancer must be configured without TLS termination.

5.9.6.2.1 In the Cloud

In cloud deployments, the load balancer is provided by the cloud provider. For example, in AWS, the service controller communicates with the AWS API to provision an ELB service which targets the cluster nodes.

5.9.6.2.2 On-premises

For an on-premises Pre-provisioned deployment, DKP ships with MetalLB[679], which provides load-balancing services. The environments that use MetalLB are pre-provisioned and vSphere infrastructures.

For more information on how to configure MetalLB for these environments, refer to the following pages:

- Pre-provisioned Configure MetalLB (see page 1093)
- Configure MetalLB for a vSphere infrastructure (see page 1376)

5.9.6.2.3 Custom Load Balancer for External Traffic

If you want to use a non-DKP load balancer for external traffic, see External Load Balancer (see page 1587).

5.9.7 Use Istio as a Microservice

Istio helps you manage cloud-based deployments by providing an open-source service mesh to connect, secure, control, and observe microservices.

This topic describes how to expose an application running on the DKP cluster using the LoadBalancer (layer-4) service type.

5.9.7.1 Prerequisites

Before you begin, verify the following:

- You must have access to a Linux, macOS, or Windows computer with a supported operating system version.
- You must have a properly deployed and running cluster.
- Determine the name of the workspace where you wish to perform the deployment. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.
- Set the `WORKSPACE_NAME` environment variable to the name of the workspace where the cluster is attached:

679 https://metallb.universe.tf/concepts/

```
export WORKSPACE_NAME=<workspace_name>
```

5.9.7.2 Deploy Istio Using DKP

Follow these steps to prepare DKP to run Istio:

1. Review the list of available applications[680] to obtain the current *APP ID* and *version* for Istio and it's dependencies (see page 673) – you need this information to execute the following commands.
2. Install Istio's dependencies (see page 673) with an `AppDeployment` resource. Replace the `<APPID>` and `<APPID-version>` variables with the information of the application:
   The required value for the `--app` flag consists of the *APP ID* and *version* in the `APPID-version` format.

   ```
   dkp create appdeployment <APPID> --app <APPID-version> --workspace $
   {WORKSPACE_NAME}
   ```

3. Install Istio. Replace `<APPID-version>` with the version of Istio you want to deploy:

   ```
   dkp create appdeployment istio --app <APPID-version> --workspace $
   {WORKSPACE_NAME}
   ```

- Create the resource in the workspace you just created, which instructs Kommander to deploy the `AppDeployment` to the `KommanderCluster`s in the same workspace.
- Observe that the `dkp create` command must be run with the `WORKSPACE_NAME` instead of the `WORKSPACE_NAMESPACE` flag.

5.9.7.2.1 Download the Istio Command Line Utility

Follow these steps to download and run Istio:

1. Pull a copy of the corresponding Istio command line to your system. Replace `<your_istio_version_here>` with the Istio version you want to deploy:

680 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/150765638/DKP+2.5.0+Components+and+Applications#applications

```
curl -L https://istio.io/downloadIstio |
ISTIO_VERSION=<your_istio_version_here> sh -
```

2. Change to the Istio directory and set your PATH environment variable by running the following commands:

```
cd istio*
export PATH=$PWD/bin:$PATH
```

3. Run the following `istioctl` command and view the subsequent output:

```
istioctl version
```

```
client version: <your_istio_version_here>
control plane version: <your_istio_version_here>
data plane version: <your_istio_version_here> (1 proxies)
```

5.9.7.2.2 Deploy a Sample Application on Istio

The Istio `bookinfo` sample application is composed of four separate microservices that demonstrate various Istio features.

1. Deploy the sample `bookinfo` application on the Kubernetes cluster by running the following commands:

   IMPORTANT: Ensure your `dkp` configuration references the cluster where you deployed Istio by setting the `KUBECONFIG` environment variable, or using the `--kubeconfig` flag, in accordance with Kubernetes conventions[681].

```
kubectl apply -f <(istioctl kube-inject -f samples/bookinfo/platform/kube/
bookinfo.yaml)
kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml
```

2. Get the URL of the load balancer created on AWS for this service by running the following command:

```
kubectl get svc istio-ingressgateway -n istio-system
```

   The command displays output similar to the following:

681 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

```
NAME                    TYPE           CLUSTER-IP     EXTERNAL-IP
PORT(S)
AGE
istio-ingressgateway    LoadBalancer   10.0.29.241
a682d13086ccf11e982140acb7ee21b7-2083182676.us-west-2.elb.amazonaws.com    15020:3
0380/TCP,80:31380/TCP,443:31390/TCP,31400:31400/TCP,15029:30756/TCP,15030:31420
/TCP,15031:31948/TCP,15032:32061/TCP,15443:31232/TCP    110s
```

3. Open a browser and navigate to the external IP address for the load balancer to access the application.

   For example, the external IP address in the sample output is `a682d13086ccf11e982140acb7ee21b7-2083182676.us-west-2.elb.amazonaws.com`, enabling you to access the application using the following URL: `http://a682d13086ccf11e982140acb7ee21b7-2083182676.us-west-2.elb.amazonaws.com/productpage`

4. Follow the steps in the Istio BookInfo Application[682] documentation to understand the different Istio features.

For more information, see Istio's official documentation[683].

5.10 GPUs

This section describes NVIDIA GPU support on Kommander. DKP supported NVIDIA driver[684] version is 470.x. GPUs, such as those made by AMD, are not currently supported. This document assumes familiarity with Kubernetes GPU support. More information about GPUs in AWS environment can be found in the Advanced AWS (see page 1249) section.

5.10.1 Kommander GPU Overview

GPU support on Kommander uses the NVIDIA[685]GPU operator. Through the NVIDIA GPU operator application, Kommander configures the container runtime to run GPU containers, and installs all the necessary items to power up the NVIDIA GPU devices.

The following components provide NVIDIA GPU support on Kommander:

- `libnvidia-container` and `nvidia-container-runtime`: GPU Support in Kommander depends on the containerd runtime. `libnvidia-container` and `nvidia-container-runtime` fit between `containerd` and `runc`, simplifying the container runtime integration with the GPU.

682 https://istio.io/docs/examples/bookinfo/
683 https://istio.io/latest/docs/
684 https://www.nvidia.com/Download/Find.aspx
685 https://github.com/NVIDIA/gpu-operator

- NVIDIA Device Plugin[686]: Kommander makes use of NVIDIA GPUs using this Kubernetes device plugin. It allows GPU enabled containers to run on Kubernetes, tracking the number of available GPUs on each node and their health.
- NVIDIA Data Center GPU Manager[687]: Contains a Prometheus exporter that provides NVIDIA GPU metrics.

Kommander runs these components as daemonsets, making them easier to manage and upgrade across all GPU nodes.

For more information from NVIDIA, see the Getting Started[688] page for NVIDIA.

5.10.2 GPU Prerequisites

5.10.2.1 Configure GPU for Kommander clusters

5.10.2.2 Prerequisites

Before you begin, you must:

- Ensure nodes provide an NVIDIA GPU.
- If you are using a public cloud service such as AWS (see page 1249), create an AMI with KIB using the instructions on the KIB for GPU (see page 1649) page.
- If you are deploying in a pre-provisioned environment, ensure that you have created the appropriate secret for your GPU nodepool (see page 239) and have uploaded the appropriate artifacts to each node. See the GPU only steps section (see page 0)on the Pre-provisioned Prerequisites Air-gapped page for additional information.

Specific instructions must be followed for enabling `nvidia-gpu-operator` depending on if you want to deploy the app on a Management cluster or a Attached or a Managed cluster.

- For instructions on enabling the NVIDIA platform application on a Management cluster, follow the instructions in the NVIDIA Platform Application Management Cluster (see page 1000) section.
- For instructions on enabling the NVIDIA platform application on attached or managed clusters, follow the instructions in the NVIDIA Platform Application Attached or Managed Cluster (see page 1002) section.

After `nvidia-gpu-operator` has been enabled depending on the cluster type, proceed to the **Select the correct Toolkit version for your NVIDIA GPU Operator** on each of those pages.

686 https://github.com/NVIDIA/k8s-device-plugin
687 https://developer.nvidia.com/dcgm
688 https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/getting-started.html#chart-customization-options

5.10.3 NVIDIA Platform Application Management Cluster

Instructions on enabling the NVIDIA platform application on a Management cluster

5.10.3.1 Enable NVIDIA Platform Application on Kommander for Management Cluster

If you intend to run applications that make use of GPU's on your cluster, you should install the NVIDIA GPU operator. To enable NVIDIA GPU support when installing Kommander on a management cluster, perform the following steps:

1. Create an installation configuration file:

   ```
   dkp install kommander --init > install.yaml
   ```

2. Append the following to the apps section in the `install.yaml` file to enable Nvidia platform services.

   ```
   apps:
     nvidia-gpu-operator:
      enabled: true
   ```

3. Install Kommander using the configuration file you created:

   ```
   dkp install kommander --installer-config ./install.yaml --kubeconfig=$
   {CLUSTER_NAME}.conf
   ```

   In the previous command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to **install Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide a context for your commands (see page 106).

4. Proceed to the Select the correct Toolkit version for your NVIDIA GPU Operator (see page 1000) section.

TIP: Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

5.10.3.2 Select the Correct Toolkit Version for your NVIDIA GPU Operator

The NVIDIA Container Toolkit allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

5.10.3.2.1 Kommander (Management Cluster) Customization

1. Select the correct Toolkit version based on your OS:

   The **NVIDIA Container Toolkit** allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

   Centos 7.9/RHEL 7.9:
   If you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

   ```
   kind: Installation
   apps:
     nvidia-gpu-operator:
      enabled: true
      values: |
        toolkit:
          version: v1.14.6-centos7
   ```

   RHEL 8.4/8.6 and SLES 15 SP3
   If you're using RHEL 8.4/8.6 or SLES 15 SP3 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

   ```
   kind: Installation
   apps:
     nvidia-gpu-operator:
      enabled: true
      values: |
        toolkit:
          version: v1.14.6-ubi8
   ```

   Ubuntu 18.04 and 20.04
   If you're using Ubuntu 18.04 or 20.04 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.14.6-ubuntu20.04
```

2. Install Kommander, using the configuration file you created:

```
dkp install kommander --installer-config ./install.yaml
```

In the previous command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to install Kommander on the right cluster. For alternatives and recommendations around setting your context, refer to Provide a context for your commands (see page 106).

TIP: Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

5.10.4 NVIDIA Platform Application Attached or Managed Cluster

Instructions on enabling the NVIDIA platform application on attached or managed clusters

5.10.4.1 Enable NVIDIA Platform Application on Attached or Managed Clusters

If you intend to run applications that utilize GPU's on Attached or Managed clusters, you must enable the `nvidia-gpu-operator` platform application in the workspace.

To use the UI to enable the application, refer to the https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919665/Platform+Applications#Customize-a-workspace%E2%80%99s-applications (see page 0) page.

To use the CLI, refer to the Deploy Platform Applications via CLI (see page 667) page.

If only a subset of attached or managed clusters in the workspace are utilizing GPU's, refer to Enable an Application per Cluster (see page 683) on how to only enable the `nvidia-gpu-operator` on specific clusters.

After you have enabled the `nvidia-gpu-operator` app in the workspace on the necessary clusters, proceed to the next section.

5.10.4.2 Select the Correct Toolkit Version for your NVIDIA GPU Operator

The NVIDIA Container Toolkit allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

5.10.4.2.1 Workspace (Attached and Managed clusters) Customization

Refer to AppDeployment resources (see page 644) for how to use the CLI to customize the platform application on a workspace.

If specific attached/managed clusters in the workspace require different configurations, refer to Customize an Application per Cluster (see page 685) for how to do this.

1. Select the correct Toolkit version based on your OS and create a `ConfigMap` with these configuration override values:

   Centos 7.9/RHEL 7.9:
   If you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set the toolkit.version parameter in your install.yaml to the following:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: nvidia-gpu-operator-overrides-attached
data:
  values.yaml: |
    toolkit:
      version: v1.10.0-centos7
EOF
```

RHEL 8.4/8.6 and SLES 15 SP3

If you're using RHEL 8.4/8.6 or SLES 15 SP3 as the base operating system for your GPU enabled nodes, set the toolkit.version parameter in your install.yaml to the following:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: nvidia-gpu-operator-overrides-attached
data:
  values.yaml: |
    toolkit:
      version: v1.10.0-ubi8
EOF
```

Ubuntu 18.04 and 20.04

If you're using Ubuntu 18.04 or 20.04 as the base operating system for your GPU enabled nodes, set the toolkit.version parameter in your install.yaml to the following:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: nvidia-gpu-operator-overrides-attached
data:
  values.yaml: |
    toolkit:
      version: v1.11.0-ubuntu20.04
EOF
```

2. Note the name of this `ConfigMap` (`nvidia-gpu-operator-overrides-attached`) and use it to set the necessary `nvidia-gpu-operator` `AppDeployment` spec fields depending on the scope of the override. Alternatively, you can also use the UI to pass in the configuration overrides for the app per workspace or per cluster.

5.10.5 Kommander GPU Settings and Troubleshoot

5.10.5.1 Validate and troubleshoot GPU

5.10.5.2 Validate that the Application has Started Correctly

Run the following command to validate that your application has started correctly:

```
kubectl get pods -A | grep nvidia
```

The output should be similar to the following:

```
nvidia-container-toolkit-daemonset-7h2l5   1/1   Running     0               150m
nvidia-container-toolkit-daemonset-mm65g   1/1   Running     0               150m
nvidia-container-toolkit-daemonset-mv7xj   1/1   Running     0               150m
nvidia-cuda-validator-pdlz8              0/1     Completed   0               150m
nvidia-cuda-validator-r7qc4              0/1     Completed   0               150m
nvidia-cuda-validator-xvtqm              0/1     Completed   0               150m
nvidia-dcgm-exporter-9r6rl               1/1     Running     1 (149m ago)    150m
nvidia-dcgm-exporter-hn6hn               1/1     Running     1 (149m ago)    150m
nvidia-dcgm-exporter-j7g7g               1/1     Running     0               150m
nvidia-dcgm-jpr57                        1/1     Running     0               150m
nvidia-dcgm-jwldh                        1/1     Running     0               150m
nvidia-dcgm-qg2vc                        1/1     Running     0               150m
nvidia-device-plugin-daemonset-2gv8h     1/1     Running     0               150m
nvidia-device-plugin-daemonset-tcmgk     1/1     Running     0               150m
nvidia-device-plugin-daemonset-vqj88     1/1     Running     0               150m
nvidia-device-plugin-validator-9xdqr     0/1     Completed   0               149m
nvidia-device-plugin-validator-jjhdr     0/1     Completed   0               149m
nvidia-device-plugin-validator-llxjk     0/1     Completed   0               149m
nvidia-operator-validator-9kzv4          1/1     Running     0               150m
nvidia-operator-validator-fvsr7          1/1     Running     0               150m
nvidia-operator-validator-qr9cj          1/1     Running     0               150m
```

If you are seeing errors, ensure that you set the container toolkit version appropriately based on your OS, as described in the previous section.

5.10.5.3 NVIDIA GPU Monitoring

Kommander uses the NVIDIA Data Center GPU Manager[689] to export GPU metrics towards Prometheus. By default, Kommander has a Grafana dashboard called `NVIDIA DCGM Exporter Dashboard` to monitor GPU metrics. This GPU dashboard is shown in Kommander's Grafana UI.

689 https://developer.nvidia.com/dcgm

5.10.5.4 NVIDIA MIG Settings

MIG stands for Multi-Instance-GPU. It is a mode of operation for future NVIDIA GPUs that allows the user to partition a GPU into a set of MIG devices. Each set appears to the software that is consuming them as a mini-GPU with a fixed partition of memory and a fixed partition of compute resources.

NOTE: MIG is only available for the following NVIDIA devices: H100, A100, and A30.

5.10.5.4.1 To Configure MIG

1. Set the MIG strategy according to your GPU topology.
   • `mig.strategy` should be set to mixed when MIG mode is not enabled on all GPUs on a node.
   • `mig.strategy` should be set to single when MIG mode is enabled on all GPUs on a node and they have the same MIG device types across all of them.

   For the Management Cluster, this can be set at install time by modifying the Kommander configuration file to add configuration for the `nvidia-gpu-operator` application:

   ```
   apiVersion: config.kommander.mesosphere.io/v1alpha1
   kind: Installation
   apps:
     nvidia-gpu-operator:
       values: |
         mig:
           strategy: single
   ...
   ```

   Or by modifying the `clusterPolicy` object for the GPU operator once it has already been installed.
2. Set the MIG profile for the GPU you are using. In our example, we are using the A30 GPU that supports the following MIG profiles:

   ```
   4 GPU instances @ 6GB each
   2 GPU instances @ 12GB each
   1 GPU instance @ 24GB
   ```

   Set the mig profile by labeling the node ${NODE} with the profile as in the example below:

```
kubectl label nodes ${NODE} nvidia.com/mig.config=all-1g.6gb --overwrite
```

3. Check the node labels to see if the changes were applied to your MIG enabled GPU node

```
kubectl get no -o json | jq .items[0].metadata.labels
```

```
"nvidia.com/mig.config": "all-1g.6gb",
  "nvidia.com/mig.config.state": "success",
  "nvidia.com/mig.strategy": "single"
```

4. Deploy a sample workload:

```
apiVersion: v1
kind: Pod
metadata:
  name: cuda-vector-add
spec:
  restartPolicy: OnFailure
  containers:
      - name: cuda-vectoradd
      image: "nvidia/samples:vectoradd-cuda11.2.1"
      resources:
        limits:
          nvidia.com/gpu: 1

  nodeSelector:
    "nvidia.com/gpu.product": NVIDIA-A30-MIG-1g.6gb
```

If the workload successfully finishes, then your GPU has been properly MIG partitioned.

5.10.5.5 Troubleshooting NVIDIA GPU Operator on Kommander

In case you run into any errors with NVIDIA GPU Operator, here are a couple commands you can run to troubleshoot:

1. Connect (using SSH or similar) to your GPU enabled nodes and run the `nvidia-smi` command. Your output should be similar to the following example:

```
[ec2-user@ip-10-0-0-241 ~]$ nvidia-smi
```

```
  Thu Nov  3 22:52:59 2022

+-----------------------------------------------------------------------------+
  | NVIDIA-SMI 470.82.01    Driver Version: 470.82.01    CUDA Version: 11.4
|
  |-------------------------------+----------------------
+----------------------+
  | GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC
|
  | Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M.
|
  |                               |                      |               MIG M.
|
  |
===============================+======================+======================|
  |   0  Tesla T4            On   | 00000000:00:1E.0 Off |                    0
 |
  | N/A   54C    P8    11W /  70W |      0MiB / 15109MiB |      0%      Default
|
  |                               |                      |                  N/A
|
  +-------------------------------+----------------------
+----------------------+

+-----------------------------------------------------------------------------+
  | Processes:
|
  |  GPU   GI   CI        PID   Type   Process name                  GPU Memory
|
  |        ID   ID                                                   Usage
|
  |
=============================================================================|
  |  No running processes found
|

+-----------------------------------------------------------------------------+
```

2. Another common issue is having a misconfigured toolkit version, resulting in NVIDIA pods in a bad state
   For example:

```
nvidia-container-toolkit-daemonset-jrqt2                          1/1
Running                 0             29s
nvidia-dcgm-exporter-b4mww                                        0/1     Error
1 (9s ago)    16s
nvidia-dcgm-pqsz8                                                 0/1
CrashLoopBackOff        1 (13s ago)   27s
nvidia-device-plugin-daemonset-7fkzr                              0/1     Init:0
/1                0             14s
```

```
nvidia-operator-validator-zxn4w                                   0/1
Init:CrashLoopBackOff   1 (7s ago)    11s
```

To modify the toolkit version, run the following commands to modify the `AppDeployment` for the `nvidia gpu operator` application:

• Provide the name of a `ConfigMap` with the custom configuration in the `AppDeployment`:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: nvidia-gpu-operator
  namespace: kommander
spec:
  appRef:
    kind: ClusterApp
    name: nvidia-gpu-operator-1.11.1
  configOverrides:
    name: nvidia-gpu-operator-overrides
EOF
```

• Create the `ConfigMap` with the name provided in the previous step, which provides the custom configuration on top of the default configuration in the config map, set the version appropriately:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: kommander
  name: nvidia-gpu-operator-overrides
data:
  values.yaml: |
    toolkit:
      version: v1.10.0-centos7
EOF
```

3. If a node has an NVIDIA GPU installed and the `nvidia-gpu-operator` application is enabled on the cluster, but the node is still not accepting GPU workloads, it's possible that the nodes do not have a label that indicates there is an NVIDIA GPU present.
By default the GPU operator will attempt to configure nodes with the following labels present, which are usually applied by the node feature discovery component:

```
    "feature.node.kubernetes.io/pci-10de.present":      "true",
```

```
    "feature.node.kubernetes.io/pci-0302_10de.present": "true",
    "feature.node.kubernetes.io/pci-0300_10de.present": "true",
```

If these labels are not present on a node that you know contains an NVIDIA GPU, you can manually label the node using the following command:

```
kubectl label node ${NODE} feature.node.kubernetes.io/
pci-0302_10de.present=true
```

5.10.5.6 Disable NVIDIA GPU Operator Platform Application on Kommander

1. Delete all GPU workloads on the GPU nodes where the NVIDIA GPU Operator platform application is present.
2. Delete the existing NVIDIA GPU Operator AppDeployment using the following command:

```
kubectl delete appdeployment -n kommander nvidia-gpu-operator
```

3. Wait for all NVIDIA related resources in the `Terminating` state to be cleaned up. You can check pod status with the following command:

```
kubectl get pods -A | grep nvidia
```

For information on how to delete nodepools, refer to Pre-provisioned Create and Delete Node Pools (see page 1142)

5.10.6 GPU Toolkit Versions

The **NVIDIA Container Toolkit** allows users to run GPU accelerated containers. The toolkit includes a container runtime library and utilities to automatically configure containers to leverage NVIDIA GPU and must be configured correctly according to your base operating system.

Centos 7.9/RHEL 7.9:
If you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.14.6-centos7
```

RHEL 8.4/8.6 and SLES 15 SP3

If you're using RHEL 8.4/8.6 or SLES 15 SP3 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.14.6-ubi8
```

Ubuntu 18.04 and 20.04

If you're using Ubuntu 18.04 or 20.04 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter in your *Kommander Installer Configuration file* (see page 1558) or `<kommander.yaml>` to the following:

```
kind: Installation
apps:
  nvidia-gpu-operator:
   enabled: true
   values: |
     toolkit:
       version: v1.14.6-ubuntu20.04
```

5.10.7 Enable GPU Usage After Installing DKP

If you want to enable your cluster to run GPU nodes after installing DKP, enable the correct Toolkit version for your operating system in the `nvidia-gpu-operator` AppDeployment.

If you have not installed the Kommander component of DKP yet, set the Toolkit version in the Kommander Installer Configuration file (see page 1010) and skip this section.

1. Create a `ConfigMap` with the necessary configuration overrides to set the correct Toolkit version (see page 1010). For example, if you're using Centos 7.9 or RHEL 7.9 as the base operating system for your GPU enabled nodes, set the `toolkit.version` parameter:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: kommander
  name: nvidia-gpu-operator-overrides
data:
  values.yaml: |
    toolkit:
      version: v1.10.0-centos7
EOF
```

   Take the correct GPU Toolkit version from this page: GPU Toolkit Versions (see page 1010)

2. Update the `nvidia-gpu-operator` AppDeployment in the `kommander` namespace to reference the `ConfigMap` you created:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: nvidia-gpu-operator
  namespace: kommander
spec:
  appRef:
    name: nvidia-gpu-operator-1.11.1
    kind: ClusterApp
  configOverrides:
    name: nvidia-gpu-operator-overrides
EOF
```

5.11 Monitoring and Alerts

Using DKP you can monitor the state of the cluster and the health and availability of the processes running on the cluster. By default, Kommander provides monitoring services using a pre-configured monitoring stack based on the Prometheus open-source project and its broader ecosystem.

The default DKP monitoring stack:

- Provides in-depth monitoring of Kubernetes components and platform services.
- Includes a default set of Grafana dashboards to visualize the status of the cluster and its platform services.

- Supports predefined critical error and warning alerts. These alerts notify immediately if there is a problem with cluster operations or availability.

By incorporating Prometheus, Kommander visualizes all the exposed metrics from your different nodes, Kubernetes objects, and platform service applications running in your cluster. The default monitoring stack also enables you to add metrics from any of your deployed applications, making those applications part of the overall Prometheus metrics stream.

5.11.1 Recommendations

Enterprise Gov Advanced

Recommended settings for monitoring and collecting metrics for Kubernetes, platform services, and applications deployed on the cluster

D2iQ conducts routine performance testing of Kommander. The following table provides recommended settings, based on cluster size and increasing workloads, that maintain a healthy Prometheus monitoring deployment.

> The resource settings reflect some settings but do not represent the exact structure to be used in the platform service configuration.

5.11.1.1 Prometheus

Cluster Size	Number of Pods	Number of Services	Resource settings

10	1k	250	<pre>resources:
			  limits:
			    cpu: 500m
			    memory:
			2192Mi
			  requests:
			    cpu: 100m
			    memory:
			500Mi
			storage: 35Gi</pre>

25	1k	250	<pre>resources:
			  limits:
			    cpu: 2
			    memory: 6Gi
			  requests:
			    cpu: 1
			    memory: 3Gi
			storage: 60Gi</pre>

50	1.5k	500	<pre>resources:
			  limits:
			    cpu: 7
			    memory:
			28Gi
			  requests:
			    cpu: 2
			    memory: 8Gi
			storage: 100Gi</pre>

100	3k	1k	<pre>resources:
			  limits:
			    cpu: 12
			    memory:
			50Gi
			  requests:
			    cpu: 10
			    memory:
			48Gi
			storage: 100Gi</pre>

200	10k	3k	<pre>resources:
			  limits:
			    cpu: 20
			    memory:
			80Gi
			  requests:
			    cpu: 15
			    memory:
			50Gi
			storage: 100Gi</pre>

300	15k	6k	<pre>resources:
			  limits:
			    cpu: 35
			    memory:
			150Gi
			  requests:
			    cpu: 25
			    memory:
			120Gi
			storage: 100Gi</pre>

5.11.2 Grafana Dashboards

With Grafana, you can query and view collected metrics in easy-to-read graphs. Kommander ships with a set of default dashboards including:

- Kubernetes Components: API Server, Nodes, Pods, Kubelet, Scheduler, StatefulSets and Persistent Volumes
- Kubernetes USE method: Cluster and Nodes
- Calico
- etcd
- Prometheus

Find the complete list of default enabled dashboards on GitHub[690].

To disable all of the default dashboards, follow these steps to define an overrides ConfigMap:

1. Create a file named `kube-prometheus-stack-overrides.yaml` and paste the following YAML code into it to create the overrides ConfigMap:

690 https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack/templates/grafana/dashboards-1.14

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-prometheus-stack-overrides
  namespace: <your-workspace-namespace>
data:
 values.yaml: |
   ---
   grafana:
     defaultDashboardsEnabled: false
```

2. Use the following command to apply the YAML file:

```
kubectl apply -f kube-prometheus-stack-overrides.yaml
```

3. Edit the `kube-prometheus-stack` AppDeployment to replace the `spec.configOverrides.name` value with `kube-prometheus-stack-overrides`. (You can use the steps in the procedure, Deploy an application with a custom configuration (see page 685) as a guide.) When your editing is complete, the AppDeployment will resemble this code sample:

```
apiVersion: apps.kommander.d2iq.io/v1alpha2
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: <your-workspace-namespace>
spec:
  appRef:
    name: kube-prometheus-stack-34.9.3
    kind: ClusterApp
  configOverrides:
    name: kube-prometheus-stack-overrides
```

To access the Grafana UI, browse to the landing page and then search for the Grafana dashboard, for example, `https://<CLUSTER_URL>/dkp/grafana`.

5.11.2.1 Add Custom Dashboards

In Kommander you can define your own custom dashboards. You can use a few methods to import dashboards[691] to Grafana.

One method is to use ConfigMaps to import dashboards[692]. Below are steps on how to create a ConfigMap with your dashboard definition.

For simplicity, this section assumes the desired dashboard definition is in `json` format:

691 https://github.com/grafana/helm-charts/tree/main/charts/grafana#import-dashboards
692 https://github.com/grafana/helm-charts/tree/main/charts/grafana#sidecar-for-dashboards

```
{
    "annotations": {
        "list": []
    },
    "description": "etcd sample Grafana dashboard with Prometheus",
    "editable": true,
    "gnetId": null,
    "hideControls": false,
    "id": 6,
    "links": [],
    "refresh": false,
    ...
}
```

After creating your custom dashboard json, insert it into a ConfigMap and save it as `etcd-custom-dashboard.yaml`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: etcd-custom-dashboard
  labels:
    grafana_dashboard: "1"
data:
  etcd.json: |
    {
        "annotations": {
            "list": []
        },
        "description": "etcd sample Grafana dashboard with Prometheus",
        "editable": true,
        "gnetId": null,
        "hideControls": false,
        "id": 6,
        "links": [],
        "refresh": false,
        ...
    }
```

Apply the ConfigMap, which automatically gets imported to Grafana using the Grafana dashboard sidecar[693]:

```
kubectl apply -f etcd-custom-dashboard.yaml
```

693 https://github.com/grafana/helm-charts/tree/main/charts/grafana#sidecar-for-dashboards

5.11.3 Cluster Metrics

The `kube-prometheus-stack` is deployed by default on the management cluster and attached clusters. This stack deploys the following Prometheus components to expose metrics from nodes, Kubernetes units, and running apps:

- prometheus-operator: orchestrates various components in the monitoring pipeline.
- prometheus: collects metrics, saves them in a time series database, and serves queries.
- alertmanager: handles alerts sent by client applications such as the Prometheus server.
- node-exporter: deployed on each node to collect the machine hardware and OS metrics.
- kube-state-metrics: simple service that listens to the Kubernetes API server and generates metrics about the state of the objects.
- grafana: monitors and visualizes metrics.
- service monitors: collects internal Kubernetes components.

> DKP has a listener on the `metrics.k8s.io/v1beta1/nodes` resource, which updates your backend store when that value changes. We then poll that backend store every 5 seconds, so the metrics are updated in realtime every 5-seconds without the need to refresh your view.

A detailed description of the exposed metrics can be found in the kube-state-metrics documentation on GitHub[694]. The `service-monitors` collect internal Kubernetes components but can also be extended to monitor customer apps as explained in the section Monitor Applications, on this page below.

5.11.4 Configure Alerts Using AlertManager

To keep your clusters and applications healthy and drive productivity forward, you need to stay informed of all events occurring in your cluster. DKP helps you to stay informed of these events by using the `alertmanager` of the `kube-prometheus-stack`.

Kommander is configured with pre-defined alerts to monitor four specific events. You receive alerts related to:

- State of your nodes
- System services managing the Kubernetes cluster
- Resource events from specific system services
- Prometheus expressions exceeding some pre-defined thresholds

Some examples of the alerts currently available are:

- CPUThrottlingHigh
- TargetDown
- KubeletNotReady

694 https://github.com/kubernetes/kube-state-metrics/tree/master/docs#exposed-metrics

- KubeAPIDown
- CoreDNSDown
- KubeVersionMismatch

A complete list with all the pre-defined alerts can be found on GitHub[695].

5.11.4.1 Prerequisites

- Determine the name of the workspace where you wish to perform the actions. You can use the `dkp get workspaces` command to see the list of workspace names and their corresponding namespaces.
- Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace where the cluster is attached:

```
export WORKSPACE_NAMESPACE=<workspace_namespace>
```

5.11.4.1.1 Configure Alert Rules

Use override `ConfigMaps` to configure alert rules.

You can enable or disable the default alert rules by providing the desired configuration in an overrides ConfigMap. For example, if you want to disable the default `node` alert rules, follow these steps to define an overrides ConfigMap:

1. Create a file named `kube-prometheus-stack-overrides.yaml` and paste the following YAML code into it to create the overrides ConfigMap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-prometheus-stack-overrides
  namespace: ${WORKSPACE_NAMESPACE}
data:
 values.yaml: |
   ---
   defaultRules:
     rules:
       node: false
```

2. Use the following command to apply the YAML file:

695 https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack/templates/prometheus/rules-1.14

```
kubectl apply -f kube-prometheus-stack-overrides.yaml
```

3. Edit the `kube-prometheus-stack` AppDeployment to replace the `spec.configOverrides.name` value with `kube-prometheus-stack-overrides`. (You can use the steps in the procedure, Deploy an application with a custom configuration (see page 1018) as a guide.)

   ```
   dkp edit appdeployment -n ${WORKSPACE_NAMESPACE} kube-prometheus-stack
   ```

   After your editing is complete, the AppDeployment resembles this example:

   ```
   apiVersion: apps.kommander.d2iq.io/v1alpha2
   kind: AppDeployment
   metadata:
     name: kube-prometheus-stack
     namespace: ${WORKSPACE_NAMESPACE}
   spec:
     appRef:
       name: kube-prometheus-stack-34.9.3
       kind: ClusterApp
     configOverrides:
       name: kube-prometheus-stack-overrides
   ```

4. To disable all rules, create an overrides ConfigMap with this YAML code:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: kube-prometheus-stack-overrides
     namespace: ${WORKSPACE_NAMESPACE}
   data:
    values.yaml: |
      ---
      defaultRules:
        create: false
   ```

5. Alert rules for the Velero platform service are turned off by default. You can enable them with the following overrides ConfigMap. They should be enabled only if the `velero` platform service is enabled. If platform services are disabled disable the alert rules to avoid alert misfires.

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: kube-prometheus-stack-overrides
   ```

```
  namespace: ${WORKSPACE_NAMESPACE}
data:
  values.yaml: |
    ---
    mesosphereResources:
      rules:
        velero: true
```

6. To create a custom alert rule named `my-rule-name`, create the overrides ConfigMap with this YAML code:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-prometheus-stack-overrides
  namespace: ${WORKSPACE_NAMESPACE}
data:
  values.yaml: |
    ---
    additionalPrometheusRulesMap:
      my-rule-name:
        groups:
        - name: my_group
          rules:
          - record: my_record
            expr: 100 * my_record
```

After you set up your alerts, you can manage each alert using the Prometheus web console to mute or unmute firing alerts, and perform other operations. For more information about configuring `alertmanager`, see the Prometheus website[696].

To access the Prometheus Alertmanager UI, browse to the landing page and then search for the Prometheus Alertmanager dashboard, for example `https://<CLUSTER_URL>/dkp/alertmanager`.

5.11.4.1.2 Notify Prometheus Alerts in Slack

To hook up the Prometheus `alertmanager` notification system, you need to overwrite the existing configuration.

1. The following file, named `alertmanager.yaml`, configures `alertmanager` to use the Incoming Webhooks feature of Slack (`slack_api_url: https://hooks.slack.com/services/<HOOK_ID>`) to fire all the alerts to a specific channel `#MY-SLACK-CHANNEL-NAME`.

```
global:
```

696 https://prometheus.io/docs/alerting/configuration/

```
  resolve_timeout: 5m
  slack_api_url: https://hooks.slack.com/services/<HOOK_ID>

route:
  group_by: ['alertname']
  group_wait: 2m
  group_interval: 5m
  repeat_interval: 1h

  # If an alert isn't caught by a route, send it to slack.
  receiver: slack_general
  routes:
    - match:
        alertname: Watchdog
      receiver: "null"

receivers:
  - name: "null"
  - name: slack_general
    slack_configs:
      - channel: '#MY-SLACK-CHANNEL-NAME'
        icon_url: https://avatars3.githubusercontent.com/u/3380462
        send_resolved: true
        color: '{{ if eq .Status "firing" }}danger{{ else }}good{{ end }}'
        title: '{{ template "slack.default.title" . }}'
        title_link: '{{ template "slack.default.titlelink" . }}'
        pretext: '{{ template "slack.default.pretext" . }}'
        text: '{{ template "slack.default.text" . }}'
        fallback: '{{ template "slack.default.fallback" . }}'
        icon_emoji: '{{ template "slack.default.iconemoji" . }}'

templates:
  - '*.tmpl'
```

2. The following file, named `notification.tmpl`, is a template that defines a pretty format for the fired notifications:

```
{{ define "__titlelink" }}
{{ .ExternalURL }}/#/alerts?receiver={{ .Receiver }}
{{ end }}

{{ define "__title" }}
[{{ .Status | toUpper }}{{ if eq .Status "firing" }}:{{ .Alerts.Firing | len }}
{{ end }}] {{ .GroupLabels.SortedPairs.Values | join " " }}
{{ end }}

{{ define "__text" }}
{{ range .Alerts }}
{{ range .Labels.SortedPairs }}*{{ .Name }}*: `{{ .Value }}`
{{ end }} {{ range .Annotations.SortedPairs }}*{{ .Name }}*: {{ .Value }}
```

```
{{ end }} *source*: {{ .GeneratorURL }}
{{ end }}
{{ end }}

{{ define "slack.default.title" }}{{ template "__title" . }}{{ end }}
{{ define "slack.default.username" }}{{ template "__alertmanager" . }}{{ end }}
{{ define "slack.default.fallback" }}{{ template "slack.default.title" . }} |
{{ template "slack.default.titlelink" . }}{{ end }}
{{ define "slack.default.pretext" }}{{ end }}
{{ define "slack.default.titlelink" }}{{ template "__titlelink" . }}{{ end }}
{{ define "slack.default.iconemoji" }}{{ end }}
{{ define "slack.default.iconurl" }}{{ end }}
{{ define "slack.default.text" }}{{ template "__text" . }}{{ end }}
```

3. Finally, apply these changes to `alertmanager` as follows. Set `${WORKSPACE_NAMESPACE}` to the workspace namespace that `kube-prometheus-stack` is deployed in:

```
kubectl create secret generic -n ${WORKSPACE_NAMESPACE} \
  alertmanager-kube-prometheus-stack-alertmanager \
  --from-file=alertmanager.yaml \
  --from-file=notification.tmpl \
  --dry-run=client --save-config -o yaml | kubectl apply -f -
```

5.11.4.1.3 Notify Prometheus Alerts in E-Mail

To configure the Prometheus `alertmanager` notification system to send an email for alerts, you need to overwrite the existing configuration. The steps below configure Alertmanager to send all configured alerts to a gmail account named test@gmail.com[697].

1. Create a file named `alertmanager.yaml` with the following contents:

```
global:
  resolve_timeout: 5m
inhibit_rules: []
receivers:
- name: "null"
- name: test_gmail
  email_configs:
  - to: test@gmail.com
    from: test@gmail.com
    auth_username: test@gmail.com
    auth_password: password
    send_resolved: true
    require_tls: true
    smarthost: smtp.gmail.com:587
```

697 mailto:test@gmail.com

```
route:
  receiver: test_gmail
  group_by:
  - namespace
  group_interval: 5m
  group_wait: 30s
  repeat_interval: 12h
  routes:
  - matchers:
    - alertname =~ "InfoInhibitor|Watchdog"
    receiver: "null"
templates:
- /etc/alertmanager/config/*.tmpl
```

2. Apply these changes to `alertmanager` as follows. Set `${WORKSPACE_NAMESPACE}` to the workspace namespace that `kube-prometheus-stack` is deployed in (typically the `kommander` namespace):

```
kubectl create secret generic -n ${WORKSPACE_NAMESPACE} \
  alertmanager-kube-prometheus-stack-alertmanager \
  --from-file=alertmanager.yaml \
  --dry-run=client --save-config -o yaml | kubectl apply -f -
```

3. Allow some time for the configuration to take affect. You can then use the following command to verify that the configuration took effect:

```
kubectl exec -it alertmanager-kube-prometheus-stack-alertmanager-0 -n kommander
-- cat /etc/alertmanager/config_out/alertmanager.env.yaml
```

For more information on configuring email alerting, refer to the Alertmanager documentation[698].

5.11.5 Centralized Monitoring

Enterprise Gov Advanced

Monitor clusters, created with Kommander, on any attached cluster

Kommander provides centralized monitoring, in a multi-cluster environment, using the monitoring stack running on any attached clusters. Centralized monitoring is provided by default in every managed or attached cluster.

Managed or attached clusters are distinguished by a monitoring ID. The monitoring ID corresponds to the kube-system namespace UID of the cluster. To find a cluster's monitoring ID, you can go to the Clusters tab on the DKP UI (in the relevant workspace), or go to the **Clusters** page in the **Global** workspace:

698 https://prometheus.io/docs/alerting/latest/configuration/

```
https://<CLUSTER_URL>/dkp/kommander/dashboard/clusters
```

Select the `View Details` link on the attached cluster card, and then select the **Configuration** tab, and find the monitoring ID under **Monitoring ID (clusterId)**.

You may also search or filter by monitoring IDs on the Clusters page, linked above.

You can also run this kubectl command, **using the correct cluster's context or kubeconfig**, to look up the cluster's kube-system namespace UID to determine which cluster the metrics and alerts correspond to:

```
kubectl get namespace kube-system -o jsonpath='{.metadata.uid}'
```

5.11.5.1 Centralized Metrics

Managed and attached clusters collects and presents metrics from all attached clusters remotely using Thanos. You can visualize these metrics in Grafana using a set of provided dashboards.

The Thanos Query[699] component is installed on attached and managed clusters. Thanos Query queries the Prometheus instances on the attached clusters, using a Thanos sidecar running alongside each Prometheus container. Grafana is configured with Thanos Query as its datasource, and comes with pre-installed dashboards for a global view of all attached clusters. The `Thanos Query` dashboard is also installed, by default, to monitor the Thanos Query component.

NOTE: Metrics from clusters are read remotely from Kommander; they are not backed up. If a attached cluster goes down, Kommander no longer collects or presents its metrics, including past data.

You can access the centralized Grafana UI at:

```
https://<CLUSTER_URL>/dkp/kommander/monitoring/grafana
```

NOTE: This is a separate Grafana instance than the one installed on all attached clusters. It is dedicated specifically to components related to centralized monitoring.

Optionally, if you want to access the Thanos Query UI (essentially the Prometheus UI), the UI is accessible at:

```
https://<CLUSTER_URL>/dkp/kommander/monitoring/query/
```

You can also check that the attached cluster's Thanos sidecars are successfully added to Thanos Query by going to:

```
https://<CLUSTER_URL>/dkp/kommander/monitoring/query/stores
```

The preferred method to view the metrics for a specific cluster is to go directly to that cluster's Grafana UI.

699 https://thanos.io/v0.5/components/query/#query

5.11.5.1.1 Adding Custom Dashboards

You can also define custom dashboards for centralized monitoring on Kommander. There are a few methods to import dashboards[700] to Grafana. For simplicity, assume the desired dashboard definition is in `json` format:

```
{
    "annotations":
    ...
    # Complete json file here
    ...
    "title": "Some Dashboard",
    "uid": "abcd1234",
    "version": 1
}
```

After creating your custom dashboard json, insert it into a ConfigMap and save it as `some-dashboard.yaml`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: some-dashboard
  labels:
    grafana_dashboard_kommander: "1"
data:
  some_dashboard.json: |
    {
      "annotations":
      ...
      # Complete json file here
      ...
      "title": "Some Dashboard",
      "uid": "abcd1234",
      "version": 1
    }
```

Apply the ConfigMap, which will automatically get imported to Grafana via the Grafana dashboard sidecar:

```
kubectl apply -f some-dashboard.yaml
```

700 https://github.com/mesosphere/charts/tree/master/stable/grafana#import-dashboards

5.11.5.2 Centralized Alerts

A centralized view of alerts, from attached clusters, is provided using an alert dashboard called Karma[701]. Karma aggregates all alerts from the Alertmanagers running in the attached clusters, allowing you to visualize these alerts on one page. Using the Karma dashboard, you can get an overview of each alert and filter by alert type, cluster, and more.

> Silencing alerts using the Karma UI is currently not supported.

You can access the Karma dashboard UI at:

```
https://<CLUSTER_URL>/dkp/kommander/monitoring/karma
```

> When there are no attached clusters, the Karma UI displays an error message `Get https://placeholder.invalid/api/v2/status: dial tcp: lookup placeholder.invalid on 10.0.0.10:53: no such host`. This is expected, and the error disappears when clusters are connected.

5.11.5.2.1 Federating Prometheus Alerting Rules

You can define additional Prometheus alerting rules[702] on attached and managed clusters and federate them to all of the attached clusters by following these instructions. To use these instructions you must install the kubefedctl CLI[703].

1. Enable the PrometheusRule type for federation.

   ```
   kubefedctl enable PrometheusRules --kubefed-namespace kommander
   ```

2. Modify the existing alertmanager configuration.

   ```
   kubectl edit PrometheusRules/kube-prometheus-stack-alertmanager.rules -n
   kommander
   ```

701 https://github.com/prymitive/karma
702 https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/
703 https://github.com/kubernetes-sigs/kubefed/blob/master/docs/installation.md#kubefedctl-cli

3. Append a sample rule.

```
- alert: MyFederatedAlert
  annotations:
    message: A custom alert that will always fire.
  expr: vector(1)
  labels:
    severity: warning
```

4. Federate the rules you just modified.

```
kubefedctl federate PrometheusRules kube-prometheus-stack-alertmanager.rules --
kubefed-namespace kommander -n kommander
```

5. Ensure that the clusters selection (`status.clusters`) is appropriately set for your desired federation strategy and check the propagation status[704].

```
kubectl get federatedprometheusrules kube-prometheus-stack-alertmanager.rules
-n kommander -oyaml
```

5.11.6 Centralized Cost Monitoring

Enterprise Gov Advanced

Monitoring costs of all attached clusters with Kubecost

Kubecost[705], running on Kommander, provides centralized cost monitoring for all attached clusters. This feature, installed by default in the management cluster, provides a centralized view of Kubernetes resources used on all attached clusters.

By default, up to 15 days of cost metrics are retained, with no backup to an external store.

704 https://github.com/kubernetes-sigs/kubefed/blob/master/docs/userguide.md#propagation-status
705 https://kubecost.com/

5.11.6.1 Adding a Kubecost License Key to your Clusters

By default, DKP is deployed with the free version of Kubecost, which provides cost monitoring for individual clusters and metric retention for up to 15 days. Licensed plans from Kubecost with additional features are also available. For more information, refer to the Kubecost Pricing[706] page.

The following instructions provide information on how you can add a Kubecost license key to your clusters if you have purchased one.

The license key must be applied to the Centralized Kubecost application running on the Management cluster.

1. Obtain license key from Kubecost. For more information, refer to the Kubecost Pricing[707] page.
2. Access the `centralized-kubecost` dashboard with the following link:

   ```
   https://<CLUSTER_URL>/dkp/kommander/kubecost/frontend/
   ```

3. From the dashboard, select the **Settings** icon, then select “Add License Key”.
   NOTE: Alternatively, the dashboard can be accessed via the following link: `https://<CLUSTER_URL>/dkp/kommander/kubecost/frontend/settings`
4. Input your license key, then select “Update.”
5. After the license key has been added, the licensed features become available in the Kubecost UI.

There is a free trial of the Kubecost Enterprise plan that allows you to preview enterprise features for 30 days. To access this, select “Start Free Trial” in the **Settings** pane.

5.11.6.1.1 Considerations when Adding a License Key

Until you add a license key, Kubecost caches the context of the cluster you first navigate from. This means that if you navigate to the Kubecost UI via the dashboard button in the Application Dashboards tab of any cluster, you will not be able to access the centralized Kubecost UI until you clear your browser cookies/ cache.

706 https://www.kubecost.com/pricing/
707 https://www.kubecost.com/pricing/

5.11.6.2 Centralized Costs

Using Thanos, the management cluster collects cost metrics remotely from each attached cluster. Costs from the last day and the last 7 days are displayed for each cluster, workspace, and project in the respective DKP UI pages. Further cost analysis and details can be found in the centralized Kubecost UI running on Kommander, at:

```
https://<CLUSTER_URL>/dkp/kommander/kubecost/frontend/detail.html#&agg=cluster
```

For more information on cost allocation metrics and how to navigate this view in the Kubecost UI, see the Kubecost docs on Kubernetes Cost Allocation[708].

To identify the clusters in Kubecost, use the cluster's monitoring ID. The monitoring ID corresponds to the kube-system namespace UID of the cluster. To find the cluster's monitoring ID, select the **Clusters** tab on the DKP UI in the relevant workspace, or go to the **Clusters** page in the **Global** workspace:

```
https://<CLUSTER_URL>/dkp/kommander/dashboard/clusters
```

Select **View Details** on the attached cluster card. Select the **Configuration** tab, and find the monitoring ID under **Monitoring ID (clusterId)**.

You can also search or filter by monitoring ID on the **Clusters** page.

To look up a cluster's kube-system namespace UID directly using the CLI, run the following kubectl command, **using the cluster's context or kubeconfig**.

```
kubectl get namespace kube-system -o jsonpath='{.metadata.uid}'
```

5.11.6.2.1 Kubecost

Kubecost integrates directly with the Kubernetes API and cloud billing APIs to give you real-time visibility into your Kubernetes spend and cost allocation. By monitoring your Kubernetes spend across clusters, you can avoid overspend caused by uncaught bugs or oversights. With a cost monitoring solution in place you can realize the full potential and cost of these resources and avoid over-provisioning resources.

To customize pricing and out of cluster costs for AWS, you must apply these settings using the Kubecost UI running on each attached cluster. You can access the attached cluster's Kubecost Settings page at:

```
https://<MANAGED_CLUSTER_URL>/dkp/kubecost/frontend/settings.html
```

Make sure you access the cluster's Kubecost UI linked above, not the centralized Kubecost UI running on the Kommander management cluster.

708 https://docs.kubecost.com/using-kubecost/getting-started/cost-allocation

5.11.6.2.1.1 AWS

For more accurate AWS Spot pricing, follow these steps[709] to configure a data feed for the AWS Spot instances.

To allocate out of cluster costs for AWS, visit this guide[710].

5.11.6.2.2 Grafana dashboards

A set of Grafana dashboards providing visualization of cost metrics is provided in the centralized Grafana UI:

```
https://<CLUSTER_URL>/dkp/kommander/monitoring/grafana
```

These dashboards give a global view of accumulated costs from all attached clusters. From the navigation in Grafana, you can find these dashboards by selecting those tagged with `cost`, `metrics`, and `utilization`.

5.11.6.3 Related Information

For information on related topics or procedures, refer to the following:

- Kubecost documentation[711]

5.11.7 Monitor Applications using Prometheus

Before attempting to monitor your own applications, you should be familiar with the Prometheus conventions for exposing metrics. In general, there are two key recommendations:

- You should expose metrics using an HTTP endpoint named `/metrics`.
- The metrics you expose must be in a format that Prometheus can consume.

By following these conventions, you ensure that your application metrics can be consumed by Prometheus itself or by any Prometheus-compatible tool that can retrieve metrics, using the Prometheus client endpoint.

The `kube-prometheus-stack` for Kubernetes provides easy monitoring definitions for Kubernetes services and deployment and management of Prometheus instances. It provides a Kubernetes resource called `ServiceMonitor`.

By default, the `kube-prometheus-stack` provides the following service monitors to collect internal Kubernetes components:

- kube-apiserver
- kube-scheduler

709 https://docs.kubecost.com/using-kubecost/getting-started/spot-checklist#implementing-spot-nodes-in-your-cluster
710 https://docs.kubecost.com/install-and-configure/advanced-configuration/cloud-integration/aws-cloud-integrations/aws-out-of-cluster
711 https://docs.kubecost.com/

- kube-controller-manager
- etcd
- kube-dns/coredns
- kube-proxy

The operator is in charge of iterating over all of these `ServiceMonitor` objects and collecting the metrics from these defined components.

The following example illustrates how to retrieve application metrics. In this example, there are:

- Three instances of a simple app named `my-app`
- The sample app listens and exposes metrics on port 8080
- The app is assumed to already be running

To prepare for monitoring of the sample app, create a service that selects the pods that have `my-app` as the value defined for their app label setting.

The service object also specifies the port on which the metrics are exposed. The `ServiceMonitor` has a label selector to select services and their underlying endpoint objects. For example:

```
kind: Service
apiVersion: v1
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    app: my-app
spec:
  selector:
    app: my-app
  ports:
  - name: metrics
    port: 8080
```

This service object is discovered by a `ServiceMonitor`, which defines the selector to match the labels with those defined in the service. The app label must have the value `my-app`.

In this example, in order for `kube-prometheus-stack` to discover this `ServiceMonitor`, add a specific label `prometheus.kommander.d2iq.io/select: "true"` in the `yaml`:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: my-app-service-monitor
  namespace: my-namespace
  labels:
    prometheus.kommander.d2iq.io/select: "true"
spec:
```

```
  selector:
    matchLabels:
      app: my-app
  endpoints:
  - port: metrics
```

In this example, you would modify the Prometheus settings to have the operator collect metrics from the service monitor by appending the following configuration to the overrides ConfigMap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-prometheus-stack-overrides
  namespace: ${WORKSPACE_NAMESPACE}
data:
  values.yaml: |
    ---
    prometheus:
      additionalServiceMonitors:
        - name: my-app-service-monitor
          selector:
            matchLabels:
              app: my-app
          namespaceSelector:
            matchNames:
              - my-namespace
          endpoints:
            - port: metrics
              interval: 30s
```

Official documentation about using a `ServiceMonitor` to monitor an app with the Prometheus-operator on Kubernetes can be found on this GitHub repository[712].

5.11.8 Set Storage Capacity for Prometheus

Follow the steps in this page to set a specific storage capacity for Prometheus.

When defining the requirements of a cluster, you can specify the capacity and resource requirements of Prometheus by modifying the settings in the overrides ConfigMap definition as shown below:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-prometheus-stack-overrides
  namespace: ${WORKSPACE_NAMESPACE}
data:
```

712 https://github.com/coreos/prometheus-operator/blob/master/Documentation/user-guides/getting-started.md#related-resources

```
values.yaml: |
  ---
  prometheus:
    prometheusSpec:
      resources:
        limits:
          cpu: "4"
          memory: "8Gi"
        requests:
          cpu: "2"
          memory: "6Gi"
    storageSpec:
      volumeClaimTemplate:
        spec:
          resources:
            requests:
              storage: "100Gi"
```

5.12 Storage for Applications

DKP ships with a Rook Ceph cluster, that is used as the primary blob storage for various DKP components in the logging stack and backups.

The pages in this section provide an overview of the Rook Ceph application in DKP, including information about its components, resource requirements, storage configuration information, and dashboard.

- Rook Ceph in DKP (see page 1034)
- BYOS (Bring Your Own Storage) to DKP Clusters (see page 1041)

5.12.1 Rook Ceph in DKP

DKP ships with a Rook Ceph cluster, that is used as the primary blob storage for various DKP components in the logging stack, backups, and DKP Insights.

The pages in this section provide an overview of the Rook Ceph application in DKP, including information about its components, resource requirements, storage configuration information, and dashboard.

The Ceph instance installed by DKP is intended only for use by DKP Insights[713] and the logging stack and `velero` platform applications.

If you have an instance of Ceph that is managed outside of the DKP lifecycle, see Bring Your Own Storage to DKP Clusters (see page 1041).

- Rook Ceph in DKP - Prerequisites (see page 1035)
- Rook Ceph Configuration (see page 1036)

713 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/62425008/DKP+Insights+Overview

- Rook Ceph Dashboard (see page 1040)

5.12.1.1 Rook Ceph in DKP - Prerequisites

Important information you should know prior to using Rook Ceph in DKP

> The Ceph instance installed by DKP is intended only for use by the logging stack and `velero` platform applications.
>
> If you have an instance of Ceph that is managed outside of the DKP lifecycle, see Bring Your Own Storage to DKP Clusters (see page 1041).

If you do not plan on using any of the logging stack components such as `grafana-loki` or `velero` for backups, then you do not need Rook Ceph for your installation and you can disable it by adding the following to your installer config file:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  ...
  ...
  grafana-loki:
    enabled: false
  ...
  rook-ceph:
    enabled: false
  rook-ceph-cluster:
    enabled: false
  ...
  velero:
    enabled: false
...
```

You must enable `rook-ceph` and `rook-ceph-cluster` if any of the following is true:

- If `grafana-loki` is enabled.
- If `velero` is enabled. If you applied config overrides for `velero` to use a storage that is external to your cluster, (see page 904)then you do not need Ceph to be installed.

> For more information about Ceph, refer to the following:

- Intro to Ceph — Ceph Documentation[714]
- Rook — Rook Ceph Documentation[715]

See Rook Ceph Configuration (see page 1036) for information on how you can configure Ceph for you Ceph Envirnoment.

5.12.1.2 Rook Ceph Configuration

This page contains information about configuring Rook Ceph in your DKP Environment.

The Ceph instance installed by DKP is intended only for use by the logging stack, DKP Insights and `velero` platform applications.

If you have an instance of Ceph that is managed outside of the DKP lifecycle, see Bring Your Own Storage to DKP Clusters (see page 1041).

If you intend to use Ceph in conjunction with DKP Insights, see DKP Insights BYOS (Bring Your Own Storage) to Insights[716].

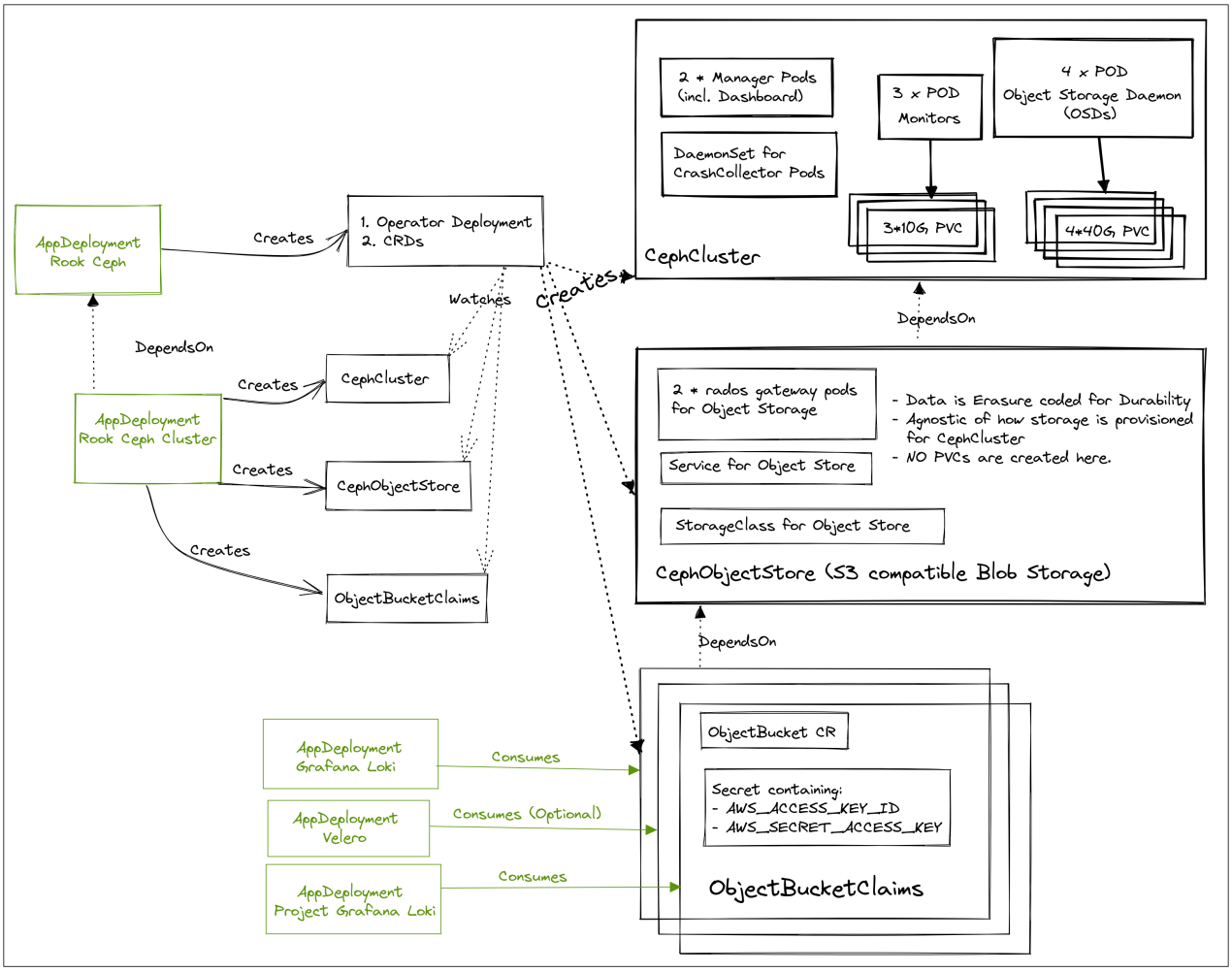
5.12.1.2.1 Components of a Rook Ceph Cluster

Ceph supports creating clusters in different modes as listed in CephCluster CRD - Rook Ceph Documentation[717]. DKP, specifically is shipped with a PVC Cluster, as documented in PVC Storage Cluster - Rook Ceph Documentation[718]. It is recommended to use the PVC mode to keep the deployment and upgrades simple and agnostic to technicalities with node draining.

Ceph cannot be your CSI Provisioner when installing in PVC mode as Ceph relies on an existing CSI provisioner to bind the PVCs created by it. It is possible to use Ceph as your CSI provisioner, but that is outside the scope of this document. If you have an instance of Ceph that acts as the CSI Provisoner, then it is possible to reuse it for your DKP Storage needs. See BYOS (Bring Your Own Storage) to DKP Clusters (see page 1041) for information on reusing existing Ceph.

When you create `AppDeployments` for `rook-ceph` and `rook-ceph-cluster` platform applications results in the deployment of various components as listed in the following diagram:

714 https://docs.ceph.com/en/quincy/start/intro/
715 https://rook.io/docs/rook/v1.10/Getting-Started/intro/
716 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/225214512
717 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/
718 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/#pvc-storage-only-for-monitors



7 Rook Ceph Cluster Components

Items highlighted in green are user-facing and configurable.

Please refer to Rook Ceph Storage Architecture[719] and Ceph Architecture[720] for an in-depth explanation of the inner workings of the components outlined in the above diagram.

For additional details about the data model, refer to the Rook Ceph Data Model[721] page.

5.12.1.2.2 Resource Requirements

The following is a non-exhaustive list of the resource requirements for long running components of Ceph:

719 https://rook.io/docs/rook/v1.10/Getting-Started/storage-architecture/
720 https://docs.ceph.com/en/quincy/architecture/
721 https://github.com/rook/rook/blob/release-1.10/design/ceph/data-model.md

Type	Resources	Total
CPUs	100m x # of mgr instances (default 2) 250m x # of mon instances (default 3) 250m x # of osd instances (default 4) 100m x # of crashcollector instances (Daemonset i.e., # of nodes) 250m x # of rados gateway replicas (default 2)	~2000m CPU
Memory	512Mi x # of mgr instances (default 2) 512Gi x # of mon instances (default 3) 1Gi x # of osd instances (default 4) 500Mi x # of rados gateway replicas (default 2)	~8Gi Memory
Disk	4 x 40Gi PVCs with `Block` mode for ObjectStorageDaemons[722] 3 x 10Gi PVCs with `Block` or `FileSystem` mode for Mons[723]	190Gi

Your default `StorageClass` should support creation of `PersistentVolume`s that satisfy the `PersistentVolumeClaim`s created by Ceph with `volumeMode: Block`.

5.12.1.2.3 Ceph Storage Configuration

Ceph is highly configurable and can support Replication or Erasure Coding to ensure data durability[724]. DKP is configured to use Erasure Coding for maximum efficiency.

Primer on Replication Strategies

Replication and Erasure Coding are the two primary methods for storing data in a durable fashion in any distributed system.

5.12.1.2.3.1 Replication[725]

- For a replication factor of N, data has N copies (including the original copy)
- Smallest possible replication factor is 2 (usually this means 2 storage nodes).
    - With replication factor of 2, data has 2 copies and this tolerates loss of one copy of data.
- Storage efficiency : `(1/N) * 100` percentage. For example,
    - If `N=2`, then efficiency is `50%`.
    - If `N=3`, then efficiency is `33%` so on.

722 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings
723 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#mon-settings
724 https://en.wikipedia.org/wiki/Durability_(database_systems)
725 https://en.wikipedia.org/wiki/Replication_(computing)

- Fault Tolerance : `N-1` nodes can be lost without loss of data. For example,
    - If `N=2` , then atmost 1 node can be lost without data loss.
    - If `N=3` , then atmost 2 nodes can be lost without data loss and so on.

5.12.1.2.3.2 Erasure Coding[726]

- Slices an object into `k` data fragments and computes `m` parity fragments. The erasure coding scheme gaurentees that data can be recreated using any `k` fragments out of `k+m` fragments.
- The `k + m = n` fragments are spread across (`>=n`) Storage Nodes to offer durability.
- Since `k` out of `n` fragments (could be parity or could be data fragments) are needed for recreation of data, at most `m` fragments can be lost without loss of data.
- The smallest possible count is `k = 2` , `m = 1` i.e., `n = k + m = 3` . This works only if there are at least `n = 3` storage nodes.
- Storage efficiency: `k/(k+m) * 100` percentage. For example,
    - If `k=2, m=1` , then efficiency is `67%`
    - If `k=3, m=1` , then efficiency is `75%` and so on.
- Fault Tolerance: `m` nodes can be lost without loss of data. For example:
    - If `k=3, m=1` then atmost 1 out of 4 nodes can be lost without data loss.
    - If `k=4, m=2` then atmost 2 out of 6 nodes can be lost without data loss and so on.

The default configuration creates a `CephCluster` that creates 4 x `PersistentVolumeClaim` s of `40G` each, resulting in `160G` of raw storage. Erasure coding ensures durability with `k=3` data bits and `m=1` parity bits. This gives a storage efficiency of `75%` (refer to the primer above for calculation), which means `120G` of disk space is available for consumption by services like `grafana-loki` , `project-grafana-loki` , and `velero` .

It is possible to override replication strategy for logging stack (`grafana-loki`) and `velero` backups. Refer to the default configmap for the `CephObjectStore` at services/rook-ceph-cluster/1.10.3/defaults/cm.yaml#L126-L175[727] and override the replication strategy according to your needs by referring to CephObjectStore CRD[728] documentation.

For more information about configuring storage in Rook Ceph, refer to the following pages:

- https://www.rook.io/docs/rook/v1.11/Storage-Configuration/Advanced/ceph-osd-mgmt/ - for general information on how to configure Object Storage Daemons (OSDs).
- https://www.rook.io/docs/rook/v1.11/Storage-Configuration/Advanced/ceph-configuration/?h=expa#auto-expansion-of-osds - for information on how to set up auto-expansion of OSDs.

---

726 https://en.wikipedia.org/wiki/Erasure_code
727 https://github.com/mesosphere/kommander-applications/blob/v2.4.0/services/rook-ceph-cluster/1.10.3/defaults/cm.yaml#L126-L175
728 https://www.rook.io/docs/rook/v1.11/CRDs/Object-Storage/ceph-object-store-crd/

See Also:

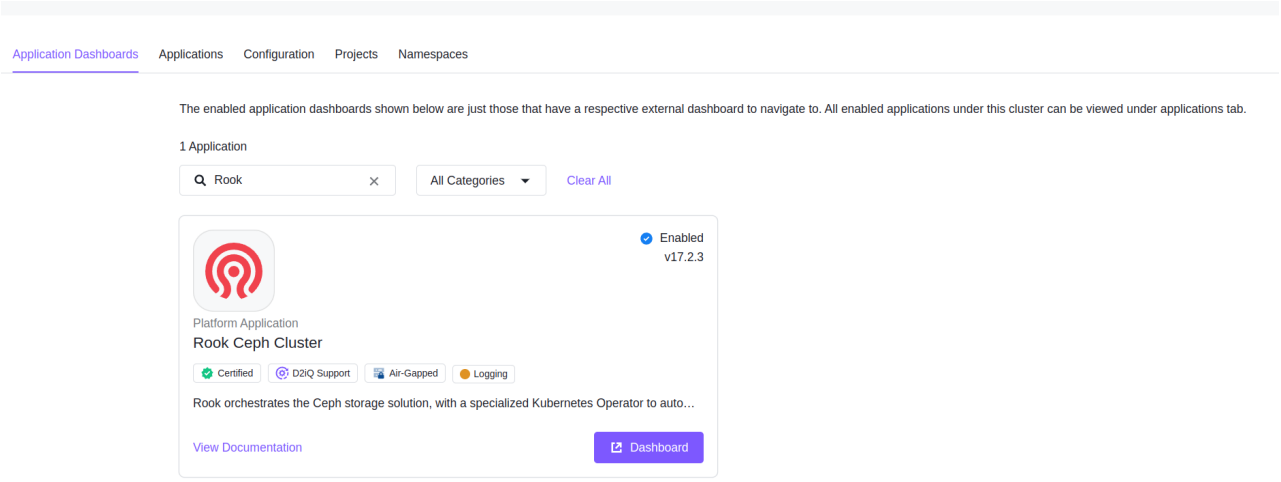
Rook Ceph in DKP - Prerequisites (see page 1035)

Rook Ceph Dashboard (see page 1040)

5.12.1.3 Rook Ceph Dashboard

5.12.1.3.1 Rook Ceph Dashboard Login

Rook Ceph Cluster provides a Dashboard which can be accessed from the UI in the Application Dashboards tab inside Kommander.



8 Rook Ceph Dashboard Link in the UI

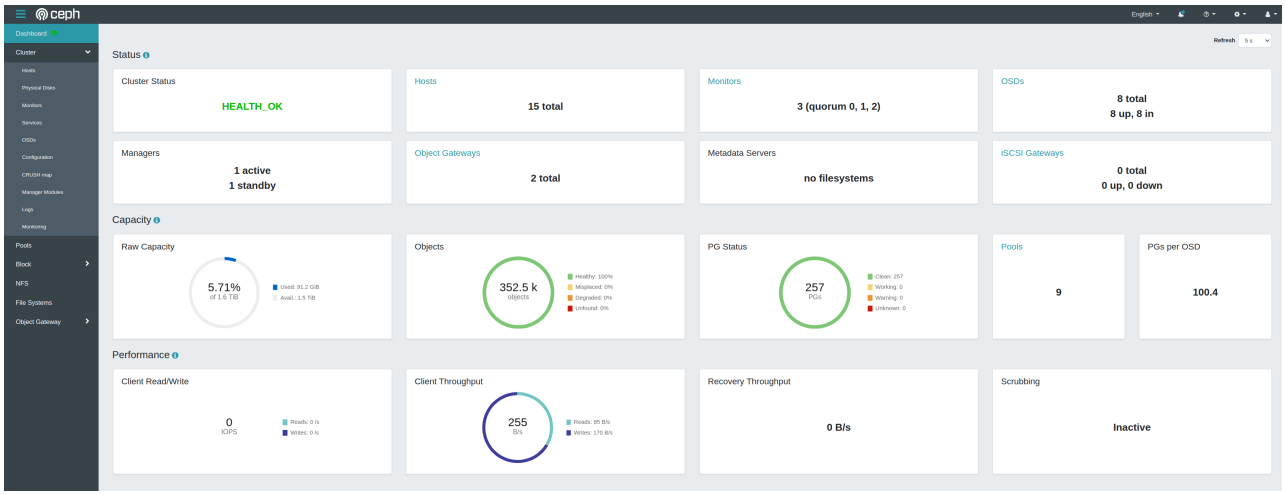
The dashboard can be used to view current cluster health and logs.

5.12.1.3.1.1 How to Access the Rook Ceph Dashboard

1. Go to the applications dashboard
2. Select the Dashboard button
3. Username is `admin`
4. To retrieve your password, in the command line **and using the kubeconfig of the Kubernetes cluster you have Rook Ceph deployed to**, run the following command:
   NOTE: Set the NAMESPACE variable according to your environment (`kommander` on management cluster or workspace namespace on attached clusters and managed clusters).

```
kubectl get secret -n ${NAMESPACE} rook-ceph-dashboard-password -ogo-
template="{{.data.password|base64decode}}"
```

5. Copy the password and paste it into the UI to access the dashboard. After successful login, a `Healthy` Rook Ceph Cluster dashboard will look similar to:



9 Rook Ceph Cluster Dashboard

5.12.2 BYOS (Bring Your Own Storage) to DKP Clusters

You can use Ceph as the CSI Provisioner in some environments. For environments where Ceph was installed before installing DKP, you can reuse your existing Ceph installation to satisfy the storage requirements of DKP Applications.

This guide assumes you have a Ceph cluster that is not managed by DKP.
Refer to Rook Ceph Configuration (see page 1036) for information on how to configure the Ceph instance installed by DKP for use by DKP platform applications.

5.12.2.1 Disable DKP Managed Ceph

Disable `rook-ceph` in your installer config since the default config of DKP has already installed a Ceph Cluster.

Disable `rook-ceph` in the installer config to prevent DKP from installing a Ceph cluster:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
```

```
apps:
  rook-ceph:
    enabled: false
  rook-ceph-cluster:
    enabled: false
  ...
...
```

The DKP instances of `velero` and `grafana-loki` rely on the storage provided by Ceph. Before installing the Kommander component of DKP, be sure to configure appropriate Ceph resources for their usage as detailed in the next section.

5.12.2.2 Creating DKP Compatible Ceph Resources

This section walks you through the creation of `CephObjectStore` and then a set of `ObjectBucketClaim`s, which can be consumed by either `velero` and `grafana-loki`.

Typically, Ceph is installed in the `rook-ceph` namespace, which is the default namespace if you have followed the Quickstart - Rook Ceph Documentation[729] guide.

> This guide assumes your Ceph instance is installed in the `rook-ceph` namespace. Configure the variable `CEPH_NAMESPACE` in subsequent steps as it is applicable to your environment.

5.12.2.2.1 Creating `CephObjectStore`

There are two ways to install Ceph:

- Using Helm Charts (see page 1041)
- Directly applying Kubernetes manifests (see page 1041)

5.12.2.2.1.1 Helm Chart Values

This section is relevant if you have installed Ceph using `helm install` or some other managed Helm resource mechanism.

If you have applied any configuration overrides to your Rook Ceph operator, ensure it was deployed with `currentNamespaceOnly` set to false[730] (It is the default value, so unless you have applied any overrides, it will be `false` by default). This ensures that the Ceph Operator in the `rook-ceph` namespace is able to monitor and manage resources in other namespaces such as `kommander`.

729 https://www.rook.io/docs/rook/v1.10/Getting-Started/quickstart/#create-a-ceph-cluster
730 https://www.rook.io/docs/rook/v1.10/Helm-Charts/operator-chart/

Ensure the following configuration[731] for `rook-ceph` helm chart[732]:

```
# This is the default value, so need to overwrite if you are just using the defaults
as-is
currentNamespaceOnly: false
```

You must enable the following configuration overrides[733] for the `rook-ceph-cluster` helm chart[734]:

```
cephObjectStores:
  - name: dkp-object-store
    # see https://github.com/rook/rook/blob/master/Documentation/CRDs/Object-Storage/
ceph-object-store-crd.md#object-store-settings for available configuration
    spec:
      metadataPool:
        # The failure domain: osd/host/(region or zone if available) - technically
also any type in the crush map
        failureDomain: osd
        # Must use replicated pool ONLY. Erasure coding is not supported.
        replicated:
          size: 3
      dataPool:
        # The failure domain: osd/host/(region or zone if available) - technically
also any type in the crush map
        failureDomain: osd
        # Data pool can use either replication OR erasure coding. Consider the
following example scenarios:
        # Erasure Coding is used here with 3 data chunks and 1 parity chunks which
assumes 4 OSDs exist.
        # Configure this according to your CephCluster specification.
        erasureCoded:
          dataChunks: 3
          codingChunks: 1
      preservePoolsOnDelete: false
      gateway:
        port: 80
        instances: 2
        priorityClassName: system-cluster-critical
        resources:
          limits:
            cpu: "750m"
            memory: "1Gi"
          requests:
            cpu: "250m"
            memory: "500Mi"
```

731 https://www.rook.io/docs/rook/v1.10/Helm-Charts/operator-chart/#configuration
732 https://www.rook.io/docs/rook/v1.10/Helm-Charts/operator-chart/#release
733 https://www.rook.io/docs/rook/v1.10/Helm-Charts/ceph-cluster-chart/#ceph-object-stores
734 https://www.rook.io/docs/rook/v1.10/Helm-Charts/ceph-cluster-chart/#release

```
    healthCheck:
      bucket:
        interval: 60s
  storageClass:
    enabled: true
    name: dkp-object-store
    reclaimPolicy: Delete
```

5.12.2.2.1.2 Managing resources directly

1. Set a variable to refer to the namespace the `AppDeployment`s are created in.

   NOTE: This is the `kommander` namespace on the management cluster or `Workspace` namespace on all other clusters.

   ```
   export CEPH_NAMESPACE=rook-ceph
   export NAMESPACE=kommander
   ```

2. Create `CephObjectStore` in the same namespace as the `CephCluster`:

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: ceph.rook.io/v1
   kind: CephObjectStore
   metadata:
     name: dkp-object-store
     namespace: ${CEPH_NAMESPACE}
   spec:
     metadataPool:
       # The failure domain: osd/host/(region or zone if available) - technically,
   any type in the crush map
       failureDomain: osd
       # Must use replicated pool ONLY. Erasure coding is not supported.
       replicated:
         size: 3
     dataPool:
       # The failure domain: osd/host/(region or zone if available) - technically,
   any type in the crush map
       failureDomain: osd
       # Data pool can use either replication OR erasure coding. Consider the
   following example scenarios:
       # Erasure Coding is used here with 3 data chunks and 1 parity chunks which
   assumes 4 OSDs exist.
       # Configure this according to your CephCluster specification.
       erasureCoded:
         dataChunks: 3
         codingChunks: 1
     preservePoolsOnDelete: false
     gateway:
   ```

```
    port: 80
    instances: 2
    priorityClassName: system-cluster-critical
    resources:
      limits:
        cpu: "750m"
        memory: "1Gi"
      requests:
        cpu: "250m"
        memory: "500Mi"
  healthCheck:
    bucket:
      interval: 60s
EOF
```

3. Wait for the `CephObjectStore` to be `Connected`:

```
$ kubectl get cephobjectstore -A
NAMESPACE   NAME               PHASE
rook-ceph   dkp-object-store   Progressing
...
...
rook-ceph   dkp-object-store   Connected
```

4. Create a `StorageClass` to consume the object storage:

```
cat <<EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: dkp-object-store
parameters:
  objectStoreName: dkp-object-store
  objectStoreNamespace: ${CEPH_NAMESPACE}
provisioner: ${CEPH_NAMESPACE}.ceph.rook.io/bucket
reclaimPolicy: Delete
volumeBindingMode: Immediate
EOF
```

5.12.2.2.2 Creating `ObjectBucketClaim`s

1. After connecting the Object Store, create the `ObjectBucketClaim` in the same namespace as `velero` and `grafana-loki`.
   This results in the creation of `ObjectBucket`, that then creates `Secret`s that are consumed by `velero` and `grafana-loki`.

a. For `grafana-loki`:

```
cat <<EOF | kubectl apply -f -
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: dkp-loki
  namespace: ${NAMESPACE}
spec:
  additionalConfig:
    maxSize: 80G
  bucketName: dkp-loki
  storageClassName: dkp-object-store
EOF
```

b. For `velero`:

```
cat <<EOF | kubectl apply -f -
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: dkp-velero
  namespace: ${NAMESPACE}
spec:
  additionalConfig:
    maxSize: 10G
  bucketName: dkp-velero
  storageClassName: dkp-object-store
EOF
```

2. Wait for the `ObjectBucket`s to be `Bound` by executing the following command:

```
kubectl get objectbucketclaim -n${NAMESPACE} -ocustom-
columns='NAME:.metadata.name,PHASE:.status.phase'
```

which should display something similar to:

```
NAME          PHASE
dkp-loki      Bound
dkp-velero    Bound
```

5.12.2.3 Configure Loki to use S3 Compatible Storage

If you wish to use your own storage in DKP that is S3 compatible, create a secret that contains your AWS secret credentials.

```
apiVersion: v1
data:
  AWS_ACCESS_KEY_ID: base64EncodedValue
  AWS_SECRET_ACCESS_KEY: base64EncodedValue
kind: Secret
metadata:
  name: dkp-loki #If you want to configure a custom name here also use it in the step
below
  namespace: kommander
```

5.12.2.4 Overriding `velero` and `grafana-loki` Configuration

After all the buckets are in the `Bound` state, DKP applications are now ready to be installed with the following configuration overrides populated in the installer config:

```
cat <<EOF | kubectl apply -f -
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  grafana-loki:
    enabled: true
    values: |
      loki:
        structuredConfig:
          storage_config:
            aws:
              s3: "http://rook-ceph-rgw-dkp-object-store.${CEPH_NAMESPACE}.svc:80/
dkp-loki"
      ingester:
        extraEnvFrom:
          - secretRef:
              name: dkp-loki # Optional: This is the default value
      querier:
        extraEnvFrom:
          - secretRef:
              name: dkp-loki # Optional: This is the default value
      queryFrontend:
        extraEnvFrom:
          - secretRef:
              name: dkp-loki # Optional: This is the default value
      compactor:
        extraEnvFrom:
          - secretRef:
              name: dkp-loki # Optional: This is the default value
      ruler:
        extraEnvFrom:
          - secretRef:
              name: dkp-loki # Optional: This is the default value
```

```
      distributor:
        extraEnvFrom:
          - secretRef:
              name: dkp-loki # Optional: This is the default value
  velero:
    enabled: true
    values: |
      configuration:
        backupStorageLocation:
        - bucket: dkp-velero
          provider: "aws"
          config:
            region: dkp-object-store
            s3Url: http://rook-ceph-rgw-dkp-object-store.${CEPH_NAMESPACE}.svc:80/
      credentials:
        # This secret is owned by the ObjectBucketClaim. A ConfigMap and a Secret
with same name as bucket are created.
        extraSecretRef: dkp-velero
EOF
```

This installer config can be merged with your installer config with any other relevant configuration before installing DKP.

5.12.2.5 Overriding `project-grafana-loki` Configuration

When installing project level grafana loki, its configuration needs to be overridden in a similar manner to workspace level grafana loki, so that the project logs can be persisted in Ceph storage.

The following overrides need to be applied to `project-grafana-loki` :

```
loki:
  structuredConfig:
    storage_config:
      aws:
        s3: "http://rook-ceph-rgw-dkp-object-store.${CEPH_NAMESPACE}.svc:80/dkp-loki"
```

These overrides can be applied from the UI directly while substituting the `${CEPH_NAMESPACE}` appropriately.

If you are using using CLI, follow these steps:

1. Set `NAMESPACE` to project namespace and `CEPH_NAMESPACE` to Ceph install namespace:

   ```
   export CEPH_NAMESPACE=rook-ceph
   export NAMESPACE=my-project
   ```

2. Create a `ConfigMap` to apply the configuration overrides:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
data:
  values.yaml: |
    loki:
      structuredConfig:
        storage_config:
          aws:
            s3: "http://rook-ceph-rgw-dkp-object-store.$
{CEPH_NAMESPACE}.svc:80/proj-loki-${NAMESPACE}"
kind: ConfigMap
metadata:
  name: project-grafana-loki-ceph
  namespace: ${NAMESPACE}
EOF
```

3. Create the `AppDeployment` with a reference to the above `ConfigMap`:
   NOTE: The `clusterSelector` can be adjusted according to your needs.

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: project-grafana-loki
  namespace: ${NAMESPACE}
spec:
  appRef:
    kind: ClusterApp
    name: project-grafana-loki-0.48.6
  clusterSelector: {}
  configOverrides:
    name: project-grafana-loki-ceph
EOF
```

The project level Grafana Loki creates an `ObjectBucketClaim` and assumes that the Ceph operator is montitoring the project namespace, so there is no need to create `ObjectBucketClaim` manually.

6 Custom Installation and Additional Infrastructure Tools

The Konvoy component of DKP can be customized for different environments and infrastructures depending on your network technology choices. If you have already installed using Day 1 - Basic Installs by Infrastructure (see page 146) instructions, you may find helpful tools in this section. However, if you have not already installed DKP with the Basic Install instructions, find your infrastructure and begin the process of custom DKP installation in this area. See the following sections for more information on custom settings:

6.1 Universal Configurations for all Infrastructure Providers

- Container Runtime Engine (CRE) (see page 1052)
- Configuring an HTTP/HTTPS Proxy (see page 1053)
- Working with Load Balancers (see page 1060)
- Customizing CAPI Components for a Cluster (see page 1061)
- Registry and Registry Mirrors (see page 1063)
- Subnets and Pods (see page 1065)
- Bastion Host (see page 1068)

6.2 Pre-provisioned Infrastructure

- Pre-provisioned Prerequisites and Environment Variables (see page 1070)
- Pre-provisioned Cluster Creation Customization Choices (see page 1083)
- Pre-provisioned Install in a Non-air-gapped Environment (see page 1105)
- Pre-provisioned Install in an Air-gapped Environment (see page 1120)
- Pre-provisioned Management Tools (see page 1138)

6.3 AWS Infrastructure

- AWS Prerequisites and Permissions (see page 1151)
- AWS Cluster Creation Customization Choices (see page 1170)
- AWS Install in a Non-air-gapped Environment (see page 1184)
- AWS Install in an Air-gapped Environment (see page 1207)
- AWS Management Tools (see page 1233)
- Configure Infrastructure in UI (see page 1258)

6.4 EKS Infrastructure

- EKS Introduction (see page 1259)
- EKS Prerequisites and Permissions (see page 1261)
- Create an EKS Cluster from the CLI (see page 1271)

6.5 Azure Infrastructure

6.6 AKS Infrastructure

6.7 vSphere Infrastructure

6.8 VMware Cloud Director Infrastructure

6.9 GCP Infrastructure

6.10 Universal Configurations for all Infrastructure Providers

Several areas of DKP configuration are common amongst all amongst all infrastructure providers. Some of the universal configurations are described in this section.

- Container Runtime Engine (CRE) (see page 1052)
- Configuring an HTTP/HTTPS Proxy (see page 1053)
- Working with Load Balancers (see page 1060)
- Customizing CAPI Components for a Cluster (see page 1061)
- Registry and Registry Mirrors (see page 1063)
- Subnets and Pods (see page 1065)
- Bastion Host (see page 1068)

6.10.1 Additional Configurations

You can find more information regarding global configurations or customization of specific components in the Additional Configurations (see page 1598) section of the documentation.

- Konvoy Image Builder (see page 1626)
- FIPS 140-2 Compliance (see page 1598)
- Air-gapped Seed the Registry (see page 1611)
- Configure the Control Plane (see page 1613)
- Update Cluster Nodepools (see page 1620)
- Kommander Customizations (see page 1558)

6.10.2 Container Runtime Engine (CRE)

A Container Runtime Engine (CRE) installed is required to install DKP. DKP supports both Podman and Docker as container engines for cluster creation. If you choose to use Podman, it works with Linux on DKP. You can choose one of these supported CREs:

- Docker®[735] container engine version 18.09.2 or higher installed for Linux or MacOS
  - On macOS, Docker runs in a virtual machine which needs to be configured with at least 8 GB of memory.
- Podman[736] version 4.0 or higher for Linux. You can find the host requirements here: Host Requirements[737].

6.10.2.1 Next Topic

Configuring an HTTP/HTTPS Proxy (see page 1053)

735 https://docs.docker.com/get-docker/
736 https://podman.io/getting-started/installation
737 https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements

6.10.3 Configuring an HTTP/HTTPS Proxy

When creating a DKP cluster in environments that use an HTTP/HTTPS proxy, you must provide proxy details. The proxy values are strings that list a set of proxy servers, URLs, or wildcard addresses that is specific to your environment.

When creating a DKP cluster in a proxied environment, you need to specify proxy settings for the:

- Bootstrap cluster
- CAPI components
- DKP Kommander component

When you create a DKP cluster using the `--self-managed` flag, the bootstrap cluster and CAPI components are created for you automatically, and use the HTTP and HTTPS proxy settings you specify in the `dkp create cluster <provider>...` command.

You can also create the bootstrap cluster and CAPI components manually, using the appropriate commands, `dkp create bootstrap` and `dkp create capi-components` respectively, combined with the command line flags to include your HTTP/S proxy information.

You can also specify HTTP/S proxy information in an override file when using Konvoy Image Builder (KIB) (see page 1670).

Without these values provided as part of the relevant `dkp create` command, DKP cannot create the requisite parts of your new cluster correctly. This is true of both management and managed clusters (see page 79) alike.

For DKP installation, you should create the bootstrap cluster from within the same network in which the new cluster will run. Using a bootstrap cluster on a laptop with different proxy settings, for example, or residing in a different network, could cause problems.

6.10.3.1 Next Steps for HTTP/HTTPS Proxy

You can define HTTP/HTTPS proxy information using the steps in these pages:

- Configure the Bootstrap Cluster HTTP Proxy Settings (see page 1054)
- Create CAPI Components with HTTP/HTTPS Proxy Settings (see page 1055)
- Create a Cluster with HTTP/HTTPS Proxy (see page 1056)
- Configure an HTTP/HTTPS Proxy for the DKP Kommander Component (see page 1058)
- Konvoy Image Builder HTTP/S Proxy (see page 1059)

6.10.3.2 Related Topic

HTTP Status Codes[738]

6.10.3.3 Next Topics

If HTTP does not apply, proceed to any of the next topics:

- Working with Load Balancers (see page 1060)
- Customizing CAPI Components for a Cluster (see page 1061)
- Registry and Registry Mirrors (see page 1063)
- Subnets and Pods (see page 1065)

6.10.3.4 Configure the Bootstrap Cluster HTTP Proxy Settings

When creating a bootstrap cluster, you must locate the device used to create the bootstrap in the same proxied environment in which the workload cluster will run. D2iQ does not recommend creating a bootstrap cluster from outside a proxied environment.

When you first install DKP, the API server doesn't exist yet in the bootstrap environment, because the API server is created *during* cluster creation. To create a bootstrap server in a proxied environment, you need to include the following flags:

- `--http-proxy <<http proxy list>>`
- `--https-proxy <<https proxy list>>`
- `--no-proxy <<no proxy list>>`

The following is an example `dkp create bootstrap` command's syntax with the HTTP proxy settings included:

```
dkp create bootstrap --http-proxy <<http proxy list>> --https-proxy <<https proxy
list>> --no-proxy <<no proxy list>>
```

6.10.3.4.1 Create a Bootstrap Cluster with HTTP Proxy Settings

Note that the delimiter between each proxy value within a flag is a comma (,) with no space character following it. The flags can include a mix of IP addresses and domain names.

1. If an HTTP proxy is required, locate the values to use for the `http_proxy`, `https_proxy`, and `no_proxy` flags. They will be built into the bootstrap cluster during cluster creation.
2. Create a bootstrap cluster using this command syntax, in addition to any other flags you may need:

738 https://github.com/kubernetes/community/blob/master/contributors/devel/sig-architecture/api-conventions.md#http-status-codes

```
dkp create bootstrap --kubeconfig $HOME/.kube/config \
  --http-proxy <string> \
  --https-proxy <string> \
  --no-proxy <string>
```

This code sample shows the command with example values for the proxy settings:

```
dkp create bootstrap \
   --http-proxy 10.0.0.15:3128 \
   --https-proxy  10.0.0.15:3128 \
   --no-proxy 127.0.0.1,192.168.0.0/16,10.0.0.0/16,10.96.0.0/12,169.254.169.254,1
69.254.0.0/24,localhost,kubernetes,kubernetes.default,kubernetes.default.svc,ku
bernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.clu
ster,.svc.cluster.local,.svc.cluster.local.,kubecost-prometheus-
server.kommander,logging-operator-logging-
fluentd.kommander.svc.cluster.local,elb.amazonaws.com
```

6.10.3.4.1.1 Next Step

Create CAPI Components with HTTP/HTTPS Proxy Settings (see page 1055)

6.10.3.5 Create CAPI Components with HTTP/HTTPS Proxy Settings

Creating CAPI components for a DKP cluster from the command line requires HTTP/HTTPS proxy information, if your environment is proxied.

If you created a cluster without using the `--self-managed flag`, the cluster will not have any of the CAPI controllers or the cert-manager component. This means that the cluster will be managed from the context of the cluster from which it was created such as the bootstrap cluster. However, you can transform the cluster to a self-managed cluster by performing the commands:
`dkp create capi-components --kubeconfig=<newcluster>`

and…
`dkp move --to-kubeconfig=<newcluster>`

This combination of actions is sometimes called a **pivot**.

When creating the CAPI components for a proxied environment using the DKP command line interface, you must include the following flags :

- `--http-proxy <<http proxy list>>`
- `--https-proxy <<https proxy list>>`
- `--no-proxy <<no proxy list>>`

The following is an example `dkp create capi-components` command's syntax with the HTTP proxy settings included:

```
dkp create capi-components --http-proxy <<http proxy list>> --https-proxy <<https
proxy list>> --no-proxy <<no proxy list>>
```

6.10.3.5.1 Create CAPI Components with HTTP Proxy Settings

Note that the delimiter between each proxy value within a flag is a comma (,) with no space character following it. The flags can include a mix of IP addresses and domain names.

1. If an HTTP proxy is required, locate the values to use for the `http_proxy`, `https_proxy`, and `no_proxy` flags. They will be built into the CAPI components during their creation.
2. Create CAPI components using this command syntax, in addition to any other flags you may need:

   ```
   dkp create capi-components --kubeconfig $HOME/.kube/config \
     --http-proxy <string> \
     --https-proxy <string> \
     --no-proxy <string>
   ```

   This code sample shows the command with example values for the proxy settings:

   ```
   dkp create capi-components \
      --http-proxy 10.0.0.15:3128 \
      --https-proxy  10.0.0.15:3128 \
      --no-proxy 127.0.0.1,192.168.0.0/16,10.0.0.0/16,10.96.0.0/12,169.254.169.254,1
   69.254.0.0/24,localhost,kubernetes,kubernetes.default,kubernetes.default.svc,ku
   bernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.clu
   ster,.svc.cluster.local,.svc.cluster.local.,kubecost-prometheus-
   server.kommander,logging-operator-logging-
   fluentd.kommander.svc.cluster.local,elb.amazonaws.com
   ```

6.10.3.5.1.1 Next Step

Create a Cluster with HTTP/HTTPS Proxy (see page 1056)

6.10.3.6 Create a Cluster with HTTP/HTTPS Proxy

During cluster creation, you may need to configure the control plane and worker nodes to use an HTTP proxy. This can occur during installation of the Konvoy component of DKP, or when creating a managed (see page 79) cluster.

If you require HTTP proxy configurations, you can apply them during the `dkp create cluster` operation by adding the appropriate flags to the command example below:

Proxy configuration	Flag
HTTP proxy for control plane machines	`--control-plane-http-proxy string`
HTTPS proxy for control plane machines	`--control-plane-https-proxy string`
No Proxy list for control plane machines	`--control-plane-no-proxy strings`
HTTP proxy for worker machines	`--worker-http-proxy string`
HTTPS proxy for worker machines	`--worker-https-proxy string`
No Proxy list for worker machines	`--worker-no-proxy strings`

You must apply the same configuration to any custom machine images built with the Konvoy Image Builder (KIB) by using an HTTP override (see page 1682) file. For more information, refer to Use Override Files with Konvoy Image Builder (see page 1677) section of the documentation.

6.10.3.6.1 Configure the Control Plane and Worker Nodes to Use HTTP/S proxy

This method for configuring the HTTP proxy values uses environment variables. (You are not required to use this method.)

Review this sample code for configuring environment variables for the control plane and worker nodes, taking into account the list of considerations that follows the sample.

```
export CONTROL_PLANE_HTTP_PROXY=http://example.org:8080
export CONTROL_PLANE_HTTPS_PROXY=http://example.org:8080
export
CONTROL_PLANE_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96
.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kubernete
s.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.clu
ster.local,169.254.169.254,.elb.amazonaws.com"

export WORKER_HTTP_PROXY=http://example.org:8080
export WORKER_HTTPS_PROXY=http://example.org:8080
export
WORKER_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96.0.0/12
,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.defau
```

```
lt.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.cluster.lo
cal,169.254.169.254,.elb.amazonaws.com"
```

HTTP proxy configuration considerations to ensure the core components work correctly

- Replace `example.org,example.com,example.net` with your internal addresses
- `localhost` and `127.0.0.1` addresses should not use the proxy
- `10.96.0.0/12` is the default Kubernetes service subnet
- `192.168.0.0/16` is the default Kubernetes pod subnet
- `kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local` is the internal Kubernetes kube-apiserver service
- The entries `.svc,.svc.cluster,.svc.cluster.local` are the internal Kubernetes services
- Auto-IP addresses `169.254.169.254` for any cloud provider

6.10.3.6.1.1 Create a Cluster Using the Configured HTTP Proxy Variables

The following is an example of a `dkp create cluster...` command that uses the values set in the environment variables from the code sample above. Use the appropriate infrastructure provider name in line 1 from the choices listed:

```
dkp create cluster [aws, azure, gcp, preprovisoned, vsphere] \
 --cluster-name ${CLUSTER_NAME} \
 --control-plane-http-proxy="${CONTROL_PLANE_HTTP_PROXY}" \
 --control-plane-https-proxy="${CONTROL_PLANE_HTTPS_PROXY}" \
 --control-plane-no-proxy="${CONTROL_PLANE_NO_PROXY}" \
 --worker-http-proxy="${WORKER_HTTP_PROXY}" \
 --worker-https-proxy="${WORKER_HTTPS_PROXY}" \
 --worker-no-proxy="${WORKER_NO_PROXY}"
```

6.10.3.6.1.2 Next Step

Configure an HTTP/HTTPS Proxy for the DKP Kommander Component (see page 1058)

6.10.3.7 Configure an HTTP/HTTPS Proxy for the DKP Kommander Component

6.10.3.7.1 HTTP Proxy for the Kommander Component of DKP

After the cluster is running in the Konvoy component from above, you need to configure the `NO_PROXY` variable for each provider.

For example, in addition to the values above for AWS, you need the following settings:

- The default VPC CIDR range of `10.0.0.0/16`
- `kube-apiserver` internal/external ELB address

- The `NO_PROXY` variable contains the Kubernetes Services CIDR. This example uses the default CIDR, `10.96.0.0/12`. If your cluster's CIDR is different, update the value in the `NO_PROXY` field.

Set the `httpProxy` and `httpsProxy` environment variables to the address of the HTTP and HTTPS proxy servers, respectively. (Frequently, environments use the same values for both.) Set the `noProxy` environment variable to the addresses that should be accessed directly, and not through the proxy.

For the Kommander component of DKP, refer to more HTTP Proxy information in Installing Kommander with an HTTP Proxy (see page 1589).

6.10.3.7.1.1 Next Step

Konvoy Image Builder HTTP/S Proxy (see page 1059)

6.10.3.8 Konvoy Image Builder HTTP/S Proxy

In some networked environments, the machines used for building images can reach the Internet, but only through an HTTP/S proxy. For DKP to operate in these networks, you need a way to specify what proxies to use (see page 1053). You can use an HTTP proxy override file to specify that proxy. When KIB is trying to install a particular OS package, it uses that proxy to reach the Internet to download that package.

The proxy setting specified here is NOT “baked into” the image - it is only used while the image is being built. The settings are removed before the image is finalized.

While it might seem logical to include the proxy information in the image, the reality is that many companies have multiple proxies - one perhaps for each geographical region, or maybe even a proxy per Datacenter or office. All network traffic to the Internet goes through the proxy. If you were in Germany, you would probably not want to send all your traffic to a U.S.-based proxy. Doing that would slow traffic down and consume too many network resources. If you baked the proxy settings into the image, you would have to create a separate image for each specific region. It makes more sense to create an image with no proxy included, but remember that you still need a proxy to access the Internet. Thus, when creating the cluster, (and also when installing the Kommander component of DKP), you must specify the correct proxy settings for the network environment into which you are installing the cluster. You will use the same base image for *that* cluster as one installed in an environment with different proxy settings.

See: HTTP Proxy Override Files (see page 1682)

6.10.3.8.1 Next Step

Either navigate to the main Konvoy Image Builder (see page 1626) section of the documentation or back to your installation section:

- If using the Day 1 - Basic Installs by Infrastructure (see page 146) instructions, proceed (or return) to that section to install and setup DKP based on your infrastructure environment provider.
- If using the Custom Installation and Additional Infrastructure Tools (see page 1050) instructions, proceed (or return) to that section and select the infrastructure provider you are using.

6.10.4 Working with Load Balancers

6.10.4.1 Selecting a Load Balancer Solution

Kubernetes comes with a basic load balancer solution internally, but Kubernetes does not directly offer an external load balancing component. You must provide one, or you can integrate your Kubernetes cluster with a cloud provider. In a Kubernetes cluster, depending on the flow of traffic direction, there are two kinds of load balancing:

- Internal load balancing for the traffic within a Kubernetes cluster
- External load balancing for the traffic coming from outside the cluster

6.10.4.1.1 External Load Balancers

DKP includes a load balancing solution for the supported cloud infrastructure providers (see page 67) and for pre-provisioned environments. For more information, see Load Balancing for external traffic (see page 994) in DKP.

If you want to use a **non-DKP load balancer** (for example, as an alternative to MetalLB in pre-provisioned environments), DKP supports setting up an **external load balancer**.

When enabled, the external load balancer routes incoming traffic requests to a single point of entry in your cluster. Users and services can then access the **DKP UI** through an established IP or DNS address.

6.10.4.1.2 Select your Connection Mechanism

A virtual IP is the address that the client uses to connect to the service. A load balancer is the device that distributes the client connections to the backend servers. Before you create a new DKP cluster, choose an external load balancer(LB) or virtual IP.

6.10.4.1.2.1 External load balancer

It is recommended that an external load balancer be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests.

6.10.4.1.2.2 Built-in virtual IP (option for Pre-provisioned or vSphere)

If an external load balancer is not available, use the built-in virtual IP (see page 1052). The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine.

6.10.4.1.3 External Load Balancer for the DKP Kommander Component

If you want to use a **non-DKP load balancer** (for example, as an alternative to MetalLB in pre-provisioned environments), DKP supports setting up an **external load balancer**.

When enabled, the external load balancer routes incoming traffic requests to a single point of entry in your cluster. Users and services can then access the **DKP UI** through an established IP or DNS address. Refer to External Load Balancer (see page 1587) page of documentation for further details.

For MetalLB, refer to the page: Pre-provisioned Configure MetalLB (see page 1093)

6.10.4.1.4 Next Topics

If Load Balancer does not apply, proceed to any of the next topics:

- Customizing CAPI Components for a Cluster (see page 1061)
- Registry and Registry Mirrors (see page 1063)
- Subnets and Pods (see page 1065)

6.10.5 Customizing CAPI Components for a Cluster

6.10.5.1 Bootstrap Cluster

Konvoy creates a bootstrap cluster using KIND[739] as a library and deploys Cluster API[740] providers on the cluster:

- Core Provider[741]
- AWS Infrastructure Provider[742]
- Kubeadm Bootstrap Provider[743]
- Kubeadm ControlPlane Provider[744]

739 https://github.com/kubernetes-sigs/kind
740 https://cluster-api.sigs.k8s.io/
741 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/
742 https://github.com/kubernetes-sigs/cluster-api-provider-aws
743 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/bootstrap/kubeadm
744 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/controlplane/kubeadm

6.10.5.2 Customizing CAPI Clusters

Familiarize yourself with Cluster API before editing the cluster objects because edits can prevent the cluster from deploying successfully. This result of this command will allow such edits.

Use the appropriate infrastructure provider name in the first line of `dkp create cluster` command from the choices listed:

```
dkp create cluster [aws, azure, gcp, preprovisoned, vsphere] \
```

EXAMPLE:

```
dkp create cluster aws
  --cluster-name=${CLUSTER_NAME} \
  --dry-run \
  --output=yaml \
  > ${CLUSTER_NAME}.yaml
```

The “>” shows the output from the command saves to the file named ${CLUSTER_NAME}.yaml. To edit that YAML, you need to understand the CAPI components to avoid failure of the cluster deployment. The objects are Custom Resources[745] defined by Cluster API components, and they belong in three different categories:

- **Cluster**
  A *Cluster* object has references to the infrastructure-specific and control plane objects. Because this is an AWS cluster, there is an *AWSCluster* object that describes the infrastructure-specific cluster properties. Here, this means the AWS region, the VPC ID, subnet IDs, and security group rules required by the Pod network implementation.
- **Control Plane**
  A *KubeadmControlPlane* object describes the control plane, which is the group of machines that run the Kubernetes control plane components. Those include the `etcd` distributed database, the API server, the core controllers, and the scheduler. The object describes the configuration for these components and has a reference to an infrastructure-specific object that describes the properties of all control plane machines. For AWS, the object references an *AWSMachineTemplate* object, which describes the instance type, the type of disk used, and the size of the disk, among other properties.
- **Node Pool**
  A Node Pool is a collection of machines with identical properties. For example, a cluster might have one Node Pool with large memory capacity and another Node Pool with GPU support. Each Node Pool is described by three objects: The MachinePool references an object that describes the configuration of Kubernetes components (for example, kubelet) deployed on each node pool machine, and an infrastructure-specific object that describes the properties of all node pool machines. For AWS, it references a *KubeadmConfigTemplate*, and an *AWSMachineTemplate* object, which describes the instance type, the type of disk used, the size of the disk, among other properties.

745 https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/

For in-depth documentation about the objects, read Concepts[746] in the Cluster API Book.

6.10.5.2.1 Next Topic

Registry and Registry Mirrors (see page 1063)

OR

Return to installation:

- If using the Day 1 - Basic Installs by Infrastructure (see page 146) instructions, proceed (or return) to that section to install and setup DKP based on your infrastructure environment provider.
- If using the Custom Installation and Additional Infrastructure Tools (see page 1050) instructions, proceed (or return) to that section and select the infrastructure provider you are using.

6.10.6 Registry and Registry Mirrors

DKP supports operation with several local registry tools (see page 1606).

- **Registry mirrors** are local copies of images from a public registry that follows (or mirrors) the file structure of a public registry. If you need to set up a private registry with a registry mirror, see this page (see page 1609) for details on using the flag(s).
- **Container registries** are collections of container repositories, and can also offer API paths and access rules.
- **Container repositories** are a collection of related container images. The container image has everything a piece of software may need to run, including code, resources, and tools. Container repositories store container images for setup and deployment, and you use the repositories to manage, pull, and push images during cluster operations.

Kubernetes does not natively provide a registry for hosting the container images you will use to run the applications you want to deploy on Kubernetes. Instead, Kubernetes requires you to use an external solution for storing and sharing container images. There are a variety of Kubernetes-compatible registry options that are compatible with DKP.

6.10.6.1 How Does it Work?

The **first time** you request an image from your local registry mirror, it pulls the image from the public registry (such as Docker) and stores it locally before handing it back to you. On subsequent requests, the local registry mirror is able to serve the image from its own storage.

6.10.6.2 Air-gapped vs Non-air-gapped Environments

In a non-air-gapped environment, you have access to the Internet. You retrieve artifacts from specialized repositories dedicated to them, such as Docker images contained in DockerHub and Helm Charts that come from a dedicated Helm Chart repository. You can also create your own local repository to hold the downloaded container images needed or any custom images you've created with the Konvoy Image Builder (see page 1626) tool.

---

746 https://cluster-api.sigs.k8s.io/user/concepts.html

In an **air-gapped environment**, you need a local repository to store Helm charts, Docker images, and other artifacts. Private registries provide security and privacy into enterprise container image storage, whether hosted remotely or on-premises locally in an air-gapped environment. DKP in an air-gapped environment **requires** a local container registry of trusted images to enable production-level Kubernetes cluster management. However, a local registry is an option in a non-air-gapped environment as well for speed and security.

If you want to use images from this local registry to deploy applications inside your Kubernetes cluster, you'll need to set up a **secret** for a private registry. The secret contains your login data, which Kubernetes needs to connect to your private repository.

It is not required to `export` any variables for most of the command examples. However, the export, along with the use of an **arbitrary** variable name, is primarily to make it clearer what values in the commands need to be substituted. Also, that makes it easier to copy and paste the examples. Furthermore, if there are multiple steps in a procedure that need you to specify a variable, you export it once with the `export`, and then reuse it in future commands.

For example,

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
```

To run the `create cluster` command using that variable created above, use the example command replacing `azure` with your choice of provider [`gcp`, `vsphere`, `vcd`, `pre-provisioned`, `aws`]:

```
dkp create cluster azure --registry-mirror-url=${REGISTRY_URL}
```

A cluster administrator uses DKP CLI commands to upload the image bundle to your registry with parameters:

```
dkp push bundle --bundle <bundle> --to-registry=${REGISTRY_URL}
```

Parameter definitions:

- `--bundle <bundle>` the group of images. The example below is for the DKP air-gapped environment bundle
- Either use exported variable `${REGISTRY_URL}` or `--to-registry=<registry-address>/<registry-name>` to provide registry location for push

An **example** command would be:

```
dkp push bundle --bundle container-images/konvoy-image-bundle-v2.8.1.tar --to-
registry=333000009999.dkr.ecr.us-west-2.amazonaws.com/can-test
```

Any URL can contain an **optional** port specification. If no port is specified, then the default port for the protocol is assumed. For example, for HTTPS protocol, port 443 is the default meaning these two URL's are equivalent:

```
https://docs.d2iq.com
https://docs.d2iq.com:443
```

A port specification is only required if the URL target is using a port number other than the default.

6.10.6.2.1 Related Topics

- Registry Mirror Tools (see page 1606)
- Use a Registry Mirror (see page 1609)
- Air-gapped Seed the Registry (see page 1611)
- Load the Images into Your Registry: Air-gapped Environments (see page 1536)

6.10.6.2.2 Next Topic

Subnets and Pods (see page 1065)

OR

Return to Installation:

- If using the Day 1 - Basic Installs by Infrastructure (see page 146) instructions, proceed (or return) to that section to install and setup DKP based on your infrastructure environment provider.
- If using the Custom Installation and Additional Infrastructure Tools (see page 1050) instructions, proceed (or return) to that section and select the infrastructure provider you are using.

6.10.7 Subnets and Pods

Some subnets are reserved by Kubernetes and can prevent proper cluster deployment if you unknowingly configure DKP so that the Node subnet collides with either the Pod or Service subnet.

Ensure your subnets do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the Kubernetes subnets, you must do this at cluster creation.

The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
```

```
    - 192.168.0.0/16
  services:
    cidrBlocks:
    - 10.96.0.0/12
```

If you need to change the Kubernetes subnets, you must do this at cluster creation. To change the subnets, perform the following steps:

1. Generate the YAML manifests for the cluster using the `--dry-run` and `-o yaml` flags, along with the desired `dkp cluster create` command:

   ```
   dkp create cluster preprovisioned --cluster-name ${CLUSTER_NAME} --control-
   plane-endpoint-host <control plane endpoint host> --control-plane-endpoint-port
   <control plane endpoint port, if different than 6443> --dry-run -o yaml >
   cluster.yaml
   ```

   ⚠ MetalLB IP address ranges/CIDRs and node subnet should not conflict with the Kubernetes cluster pod and service subnets.

2. To modify the service subnet, add or edit the `spec.clusterNetwork.services.cidrBlocks` field of the `Cluster` object:

   ```
   kind: Cluster
   spec:
     clusterNetwork:
       services:
         cidrBlocks:
         - 10.0.0.0/12
   ```

3. To modify the pod subnet, edit the `Cluster` and calico-cni `ConfigMap` resources:

   Cluster: Add or edit the `spec.clusterNetwork.pods.cidrBlocks` field:

   ```
   kind: Cluster
   spec:
     clusterNetwork:
       pods:
         cidrBlocks:
         - 172.16.0.0/16
   ```

   ConfigMap: Edit the `data."custom-resources.yaml".spec.calicoNetwork.ipPools.cidr` field with your desired pod subnet:

```
apiVersion: v1
data:
  custom-resources.yaml: |
    apiVersion: operator.tigera.io/v1
    kind: Installation
    metadata:
      name: default
    spec:
      # Configures Calico networking.
      calicoNetwork:
        # Note: The ipPools section cannot be modified post-install.
        ipPools:
        - blockSize: 26
          cidr: 172.16.0.0/16
kind: ConfigMap
metadata:
  name: calico-cni-<cluter-name>
```

When you provision the cluster, the configured pod and service subnets will be applied.

6.10.7.1 Related Topics

- Pre-provisioned Configure MetalLB (see page 1093)
- Configure MetalLB for a vSphere infrastructure (see page 1376)

6.10.7.2 Next Steps

Proceed to installation instructions:

- If using the Custom Installation and Additional Infrastructure Tools (see page 1050) instructions, proceed to the infrastructure provider you are using.
    - Pre-provisioned Infrastructure (see page 1070)
    - AWS Infrastructure (see page 1148)
    - EKS Infrastructure (see page 1259)
    - Azure Infrastructure (see page 1296)
    - AKS Infrastructure (see page 1343)
    - vSphere Infrastructure (see page 1354)
    - VMware Cloud Director Infrastructure (see page 1444)
    - GCP Infrastructure (see page 1486)
- If using the Day 1 - Basic Installs by Infrastructure (see page 146) instructions, proceed (or return) to that section to install and setup DKP based on your infrastructure environment provider.

6.10.8 Bastion Host

When creating an air-gapped cluster, the bastion VM hosts the installation of the DKP Konvoy bundles and images, as well as the Docker or other local registry (see page 1606), needed to create and operate your cluster. In a given environment, the bastion VM must have access to the infrastructure provider's API. Ensure the items below are installed and the environment matches the requirements below:

- Create a bastion VM host template for the cluster nodes to use within the air-gapped network. This bastion VM host also needs access to a local registry in lieu of an Internet connection for pulling images.
- Find and record the bastion VM's IP or host name.
- Download (see page 76) the following required DKP Konvoy binaries and installation bundles discussed in step 5 below.
- A local registry (see page 1606) or Docker®[747] version 18.09.2 or later installed. You must have Docker installed on the host where the DKP Konvoy CLI runs. For example, if you are installing Konvoy on your laptop, ensure the laptop has a supported version of Docker. On macOS, Docker runs in a virtual machine which you configure with at least 8GB of memory.
- kubectl[748] for interacting with the running cluster, installed on the host where the DKP Konvoy command line interface (CLI) runs.

Depending on your OS, there are various commands for setting up your own bastion host for use in an air-gapped environment. The steps below are an example for vSphere.

This would be a **generic** example for RHEL Bastion nodes using Docker:

1. Open an `ssh` terminal to the bastion host and install the tools and packages:

   ```
   sudo yum install -y yum-utils bzip2 wget
   ```

2. Install kubectl as mentioned above, below is a RHEL example:

   ```
   cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
   [kubernetes]
   name=Kubernetes
   baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch
   enabled=1
   gpgcheck=1
   gpgkey=https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
   EOF
   sudo yum install -y kubectl
   ```

3. Install Docker for example (only on the Bastion Host) and add the repo for upstream Docker:

747 https://docs.docker.com/get-docker/
748 https://kubernetes.io/docs/tasks/tools/#kubectl

```
sudo yum-config-manager --add-repo https://download.docker.com/linux/rhel/
docker-ce.repo
```

NOTE: Other Docker repo downloads are available on docker.com[749]: https://download.docker.com/linux/

4. Install example for Docker:

```
sudo yum install -y docker-ce docker-ce-cli containerd.io
```

5. Get the needed D2iQ Software by downloading the air-gapped bundle:
Download (see page 76) `dkp-air-gapped-bundle_v2.7.0_linux_amd64.tar.gz`, extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.7.0_linux_amd64.tar.gz
```

6. Set the following environment variables to enable connection to an existing Docker or other registry. You must create the VM template with the Konvoy Image Builder (see page 1626) to be able to use the registry mirror feature:

```
export REGISTRY_ADDRESS=<https/http>://<registry-address>:<registry-port>
export REGISTRY_CA=<path to the CA on the bastion host>
```

- `REGISTRY_ADDRESS`: the address of an existing registry accessible in the environment where the new cluster nodes will be configured, to use a mirror registry when pulling images.
- `REGISTRY_CA`: (optional) the path on the bastion host to the registry CA. Konvoy configures the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the VMs are not already configured to trust this CA.

6.10.8.1 More information:

Each infrastructure provider has its own set of bastion host instructions. Refer to your own OS instructions to setup a bastion host like AWS Bastion[750], Azure[751], GCP[752], or vSphere[753].

6.10.8.2 Next Step

Proceed to installation instructions:

749 http://docker.com
750 https://aws.amazon.com/solutions/implementations/linux-bastion/
751 https://learn.microsoft.com/en-us/azure/bastion/quickstart-host-portal
752 https://blogs.vmware.com/cloud/2021/06/02/intro-google-cloud-vmware-engine-bastion-host-access-iap/
753 https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.security.doc/GUID-6975426F-56D0-4FE2-8A58-580B40D2F667.html

- If using the Custom Installation and Additional Infrastructure Tools (see page 1050) instructions, proceed to the infrastructure provider you are using.
    - Pre-provisioned Infrastructure (see page 1070)
    - AWS Infrastructure (see page 1148)
    - EKS Infrastructure (see page 1259)
    - Azure Infrastructure (see page 1296)
    - AKS Infrastructure (see page 1343)
    - vSphere Infrastructure (see page 1354)
    - VMware Cloud Director Infrastructure (see page 1444)
    - GCP Infrastructure (see page 1486)
- If using the Day 1 - Basic Installs by Infrastructure (see page 146) instructions, proceed (or return) to that section to install and setup DKP based on your infrastructure environment provider.

6.11 Pre-provisioned Infrastructure

6.11.1 Create a Kubernetes cluster on pre-provisioned nodes in a bare metal infrastructure

The following procedure describes creating a DKP cluster on a pre-provisioned infrastructure using SSH. For a more detailed definition of a Pre-provisioned environment, refer to What is Pre-provisioned Infrastructure (see page 84).

Completing this procedure results in a Kubernetes cluster that includes a Container Networking Interface (CNI)[754] and a Local Persistence Volume Static Provisioner[755], and that is ready for workload deployment.

Before moving to a production environment, you may want to add applications for logging and monitoring, storage, security, and other functions. You can use DKP to select and deploy applications (see page 667), or deploy your own.

To **get started**, see these configuration and installation topics:

- Pre-provisioned Prerequisites and Environment Variables (see page 1070)
- Pre-provisioned Cluster Creation Customization Choices (see page 1083)
- Pre-provisioned Install in a Non-air-gapped Environment (see page 1105)
- Pre-provisioned Install in an Air-gapped Environment (see page 1120)
- Pre-provisioned Management Tools (see page 1138)

6.11.2 Pre-provisioned Prerequisites and Environment Variables

Pre-provisioning is the process of setting up an environment that authorized users, devices, and servers can access. In practice, network provisioning primarily concerns connectivity and security, which means a heavy

754 https://docs.projectcalico.org/
755 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner

focus on device and identity management. Pre-provisioning can bring enterprises greater efficiency and more secure operations.

To function on a network, a server, whether cloud-based or on-premises, must first be provisioned with the right data, software, and configuration.

The steps in this process typically include:

- Installing an operating system, device drivers, and partitioning and setup tools
- Installing enterprise software and applications
- Setting parameters such as IP addresses
- Performing partitioning or installation of virtualization software
- Connectivity whether an air-gapped or non-air-gapped environment meaning it is connected to the internet

Environment prerequisites and configuration is found on these pages before install begins:

- Pre-provisioned Prerequisite Configuration (see page 1071)
- Pre-provisioned Set Infrastructure (see page 1074)
- Pre-provisioned Loading the Registry (see page 1076)
- Pre-provisioned Azure only Configurations (see page 1079)

6.11.2.1 Next Step

Pre-provisioned Prerequisite Configuration (see page 1071)

6.11.2.2 Pre-provisioned Prerequisite Configuration

In order to fulfill all the prerequisites for a successful implementation on a Pre-provisioned environment, there will be infrastructure requirements as well as machine requirements. Please read all the sections on this page to ensure you have met all prerequisites.

6.11.2.2.1 Prerequisites for a Pre-provisioned Infrastructure

Before you begin using DKP, you must have:

- An x86_64-based Linux or macOS machine.
- The `dkp` binary for Linux, or macOS.
- kubectl[756] for interacting with the running cluster.
- Pre-provisioned hosts with SSH access enabled.
- An unencrypted SSH private key, whose public key is configured on the above hosts.
- A Container engine/runtime installed is required to install DKP:
    - Version Docker®[757] container engine version 18.09.2 or higher installed for Linux or MacOS - On macOS, Docker runs in a virtual machine which needs configured with at least 8 GB of memory.

756 https://kubernetes.io/docs/tasks/tools/#kubectl
757 https://docs.docker.com/get-docker/

    - Version 4.0 of Podman[758] or higher for Linux. Host requirements found here: Host Requirements[759]
- To use a local registry whether air-gapped or non-air-gapped environment, download and extract the bundle. Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz`) to load registry.
- For an air-gapped environment, create a working registry:
    - local registry (see page 1606) on bastion or other machine
- Resource requirements (see page 119)
- Pre-provisioned Override Files (see page 1680)

When in an air-gapped environment, you must follow the steps described in the Air-gapped Define Environment (see page 1120) and Docker Registry also as a prerequisite.

DKP uses `localvolumeprovisioner` as the default storage provider (see page 110). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[760] compatible storage that is suitable for production.

You can choose from any of the storage options[761] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[762].

6.11.2.2.1.1 Machine Specifications

Control plane machines

You should have at least three control plane machines.

Each control plane machine must have:

- 4 cores
- 16 GiB memory
- Approximately 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.
- 15% free space on the root file system.
- Multiple ports open, as described in DKP Ports (see page 65).

758 https://podman.io/getting-started/installation
759 https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
760 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
761 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
762 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

- `firewalld` systemd service disabled. If it exists and is enabled, use the commands `systemctl stop firewalld` then `systemctl disable firewalld`, so that `firewalld` remains disabled after the machine restarts.
- For a Pre-provisioned environment using Ubuntu 20.04, ensure the machine has the `/run` directory mounted with exec permissions.

> Swap is disabled. The `kubelet` does not have generally-available support for swap. Due to variable commands, refer to your operating system documentation.

Worker machines

You should have at least four worker machines. The specific number of worker machines required for your environment can vary depending on the cluster workload and size of the machines.

Each worker machine must have:

- 8 cores
- 32 GiB memory
- Around 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.
- 15% free space on the root file system
- If you plan to use local volume provisioning to provide persistent volumes for your workloads, you must mount at least four volumes to the `/mnt/disks/` mount point on each machine. Each volume must have at least 55 GiB of capacity.
- Ensure your disk meets the resource requirements for Rook Ceph in `Block` mode for ObjectStorageDaemons[763] as specified in the requirements table (see page 1036).
- Multiple ports open, as described in DKP Ports (see page 65).
- `firewalld` systemd service disabled. If it exists and is enabled, use the commands `systemctl stop firewalld` then `systemctl disable firewalld`, so that `firewalld` remains disabled after the machine restarts.
- For a Pre-provisioned environment using Ubuntu 20.04, ensure the machine has the `/run` directory mounted with exec permissions.

> Swap is disabled. The `kubelet` does not have generally-available support for swap. Due to variable commands, refer to your operating system documentation.

763 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

6.11.2.2.1.2 Next Step

Pre-provisioned Set Infrastructure (see page 1074)

6.11.2.3 Pre-provisioned Set Infrastructure

The Konvoy component of DKP needs to know how to access your cluster hosts so you must define the cluster hosts and infrastructure. This is done using inventory resources. For initial cluster creation, you must define a control-plane and at least one worker pool for both air-gapped and non-air-gapped environments.

6.11.2.3.1 Define your Infrastructure

1. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CLUSTER_NAME=<preprov cluster name>
export CONTROL_PLANE_1_ADDRESS=<control-plane-address-1>
export CONTROL_PLANE_2_ADDRESS=<control-plane-address-2>
export CONTROL_PLANE_3_ADDRESS=<control-plane-address-3>
export WORKER_1_ADDRESS=<worker-address-1>
export WORKER_2_ADDRESS=<worker-address-2>
export WORKER_3_ADDRESS=<worker-address-3>
export WORKER_4_ADDRESS=<worker-address-4>
export SSH_USER=<ssh-user>
export SSH_PRIVATE_KEY_SECRET_NAME=$CLUSTER_NAME-ssh-key
```

2. Use the following template to help you define your infrastructure. The environment variables that you set in the previous step automatically replace the variable names when the file is created.

```
cat <<EOF > preprovisioned_inventory.yaml
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: ${CLUSTER_NAME}-control-plane
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: ${CLUSTER_NAME}
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    # Create as many of these as needed to match your infrastructure
    # Note that the command line parameter --control-plane-replicas determines
how many control plane nodes will actually be used.
    #
    - address: ${CONTROL_PLANE_1_ADDRESS}
```

```
    - address: ${CONTROL_PLANE_2_ADDRESS}
    - address: ${CONTROL_PLANE_3_ADDRESS}
  sshConfig:
    port: 22
    # This is the username used to connect to your infrastructure. This user
must be root or
    # have the ability to use sudo without a password
    user: $SSH_USER
    privateKeyRef:
      # This is the name of the secret you created in the previous step. It
must exist in the same
      # namespace as this inventory object.
      name: ${SSH_PRIVATE_KEY_SECRET_NAME}
      namespace: default
---
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: ${CLUSTER_NAME}-md-0
  namespace: default
  labels:
    cluster.x-k8s.io/cluster-name: ${CLUSTER_NAME}
    clusterctl.cluster.x-k8s.io/move: ""
spec:
  hosts:
    - address: ${WORKER_1_ADDRESS}
    - address: ${WORKER_2_ADDRESS}
    - address: ${WORKER_3_ADDRESS}
    - address: ${WORKER_4_ADDRESS}
  sshConfig:
    port: 22
    user: ${SSH_USER}
    privateKeyRef:
      name: ${SSH_PRIVATE_KEY_SECRET_NAME}
      namespace: default
EOF
```

3. To tell the bootstrap cluster which nodes you want to be control plane nodes and which nodes are worker nodes, use the `kubectl apply` command to apply the file to the bootstrap cluster:

```
kubectl apply -f preprovisioned_inventory.yaml
```

Output:

```
preprovisionedinventory.infrastructure.cluster.konvoy.d2iq.io/preprovisioned-
example-control-plane created
preprovisionedinventory.infrastructure.cluster.konvoy.d2iq.io/preprovisioned-
example-md-0 created
```

6.11.2.3.1.1 Next Step

Pre-provisioned Cluster Creation Customization Choices (see page 1083)

If applicable for a local registry or special Azure considerations, refer to these sections also:

- Pre-provisioned Loading the Registry (see page 1076)
- Pre-provisioned Azure only Configurations (see page 1079)

6.11.2.4 Pre-provisioned Loading the Registry

The complete DKP air-gapped bundle is needed for an air-gapped environment but can also be used in a non-air-gapped environment. The bundle contains all the DKP components needed for an air-gapped environment installation and also to use a local registry in a non-air-gapped environment.

6.11.2.4.1 Download all Images for Air-gapped Deployments

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for prerequisites to download and then how to push the necessary images to this registry.

1. Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`) to load registry images as explained below.
2. Connectivity with clusters attaching to the management cluster is required:
   - Both management and attached clusters must be able to connect to the local registry.
   - The management cluster must be able to connect to all attached cluster's API servers.
   - The management cluster must be able to connect to any load balancers created for platform services on the management cluster.

6.11.2.4.2 Extract Air-gapped Images

Follow these steps to **extract** the air-gapped image bundles into your private registry.

6.11.2.4.2.1 Load the Bootstrap Image

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz` from the download site (see page 76) mentioned above, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz
   ```

EXAMPLE directory structure after extraction:

```
dkp-v2.8.0/
├── application-charts
│   ├── NOTICES.txt
│   ├── dkp-catalog-applications-charts-bundle-v2.8.0-rc.2.tar.gz
│   ├── dkp-insights-charts-bundle-v2.8.0-rc.2.tar.gz
│   └── dkp-kommander-charts-bundle-v2.8.0-rc.2.tar.gz
├── application-repositories
│   ├── dkp-catalog-applications-v2.8.0-rc.2.tar.gz
│   ├── dkp-insights-v2.8.0-rc.2.tar.gz
│   └── kommander-applications-v2.8.0-rc.2.tar.gz
├── container-images
│   ├── NOTICES.txt
│   ├── dkp-catalog-applications-image-bundle-v2.8.0-rc.2.tar
│   ├── dkp-insights-image-bundle-v2.8.0-rc.2.tar
│   ├── kommander-image-bundle-v2.8.0-rc.2.tar
│   └── konvoy-image-bundle-v2.8.0-rc.2.tar
├── dkp
├── kib
│   ├── LICENSE
│   ├── README.md
│   ├── ansible
│   ├── artifacts
│   ├── goss
│   ├── images
│   ├── konvoy-image
│   └── overrides
├── konvoy-bootstrap-image-v2.8.0-rc.2.tar
└── kubectl
```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. For the bootstrap, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

   ```
   cd dkp-v2.8.0
   ```

3. Load the bootstrap container image on your bastion machine replacing `docker` with `podman` if needed:

   ```
   docker load -i konvoy-bootstrap-image-v2.8.0.tar
   ```

6.11.2.4.2.2 Set Environment Variables

1. Set an environment variable with your registry address:

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
```

- `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.

More Registry Variables:

More environment variables if needed:

```
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
export REGISTRY_CA=<path to the cacert file on the bastion>
```

6.11.2.4.2.3 Load Images to your Private Registry - Konvoy

Before creating or upgrading a Kubernetes cluster, you need to load the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, refer to Local Registry Tools (see page 1606) page for more information.

Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

6.11.2.4.2.4 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

6.11.2.4.2.5 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME}
--to-registry-password=${REGISTRY_PASSWORD}
```

6.11.2.4.2.6 Next Step

If applicable for special Azure considerations, refer to this section also:

- Pre-provisioned Azure only Configurations (see page 1079)

Otherwise, proceed to Pre-provisioned Cluster Creation Customization Choices (see page 1083)

6.11.2.5 Pre-provisioned Azure only Configurations

After your bootstrap is running and your cluster is created, you will need to install the Azure Disk CSI Driver[764] on your pre-provisioned Azure Kubernetes cluster. The DKP pre-provisioned provider installs by default the storage-local-static-provisioner[765] CSI driver, which is not suitable for production environments. For this reason, it needs to be replaced by the Azure Disk CSI Driver[766].

6.11.2.5.1 Prerequisites:

Before you begin using DKP you must have:

- An x86_64-based Linux or macOS machine.
- Download (see page 76) the `dkp` binary for Linux, or macOS. To check which version of DKP you installed for compatibility reasons, run the `dkp version` command (dkp version (see page 1943)).
- A container engine:

764 https://github.com/kubernetes-sigs/azuredisk-csi-driver/tree/master
765 https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner
766 https://github.com/kubernetes-sigs/azuredisk-csi-driver/blob/master/docs/install-azuredisk-csi-driver.md

    - Version 4.0 of Podman[767] or higher for Linux
    - Version 18.09.2 of Docker[768] or higher for Linux or MacOS
- kubectl[769] for interacting with the running cluster.
- Azure CLI[770].
- A valid Azure account with credentials configured[771].
- Create a custom Azure image using KIB (see page 1642).
- **For air-gapped environments only** -
    - Ability to download artifacts from the internet and then copy those onto your bastion machine.
    - Download (see page 76) the Complete DKP Air-gapped Bundle for this release - `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz`.
    - An existing local registry (see page 1606) to seed the air-gapped environment.

On macOS, Docker runs in a virtual machine. Configure this virtual machine with at least 8GB of memory.

6.11.2.5.2 Set Environment Variables with Credentials:

An Azure Service Principal is needed for deploying resources. To configure your Azure environment[772], follow below:

1. Log in to Azure:

```
az login
```


```
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "id": "b1234567-abcd-11a1-a0a0-1234a5678b90",
```


767 https://podman.io/getting-started/installation
768 https://www.docker.com/
769 https://kubernetes.io/docs/tasks/tools/#kubectl
770 https://docs.microsoft.com/en-us/cli/azure/install-azure-cli
771 https://github.com/kubernetes-sigs/cluster-api-provider-azure/blob/master/docs/book/src/topics/getting-started.md#prerequisites
772 https://github.com/kubernetes-sigs/cluster-api-provider-azure/blob/main/docs/book/src/topics/getting-started.md#setting-up-your-azure-environment

```
    "isDefault": true,
    "managedByTenants": [],
    "name": "Mesosphere Developer Subscription",
    "state": "Enabled",
    "tenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "user": {
      "name": "user@azuremesosphere.onmicrosoft.com",
      "type": "user"
    }
  }
]
```

2. Create an Azure Service Principal (SP) by running the following command:
   Note: If an SP with the name exists, this command will rotate the password.

```
az ad sp create-for-rbac --role contributor --name "$(whoami)-konvoy" --
scopes=/subscriptions/$(az account show --query id -o tsv)
```

```
{
  "appId": "7654321a-1a23-567b-b789-0987b6543a21",
  "displayName": "azure-cli-2021-03-09-23-17-06",
  "password": "Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C",
  "tenant": "a1234567-b132-1234-1a11-1234a5678b90"
}
```

   For air-gapped environments, you need to create a resource management private link[773] with a private endpoint to ensure the Azure CSI driver will run correctly in further steps. Private links enable you to access Azure services over a private endpoint in your virtual network.
   To set up a private link resource, use the following process.
   a. Create the resource management private link[774] using Azure CLI.
   b. Create a private link association[775] for the root management group which also references the resource ID for the resource management private link.
   c. Add a private endpoint that references the resource management private link using the Azure Documentation[776].
3. Set the required environment variables using that output:

773 https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/create-private-link-access-portal
774 https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/create-private-link-access-commands?tabs=azure-cli#create-resource-management-private-link
775 https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/create-private-link-access-commands?tabs=azure-cli#create-private-link-association
776 https://learn.microsoft.com/en-us/azure/private-link/create-private-endpoint-cli?tabs=dynamic-ip

```
export AZURE_SUBSCRIPTION_ID="<id>"       # b1234567-abcd-11a1-
a0a0-1234a5678b90
export AZURE_TENANT_ID="<tenant>"         # a1234567-b132-1234-1a11-1234a5678b9
0
export AZURE_CLIENT_ID="<appId>"          # 7654321a-1a23-567b-
b789-0987b6543a21
export AZURE_CLIENT_SECRET="<password>"   # Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C
export AZURE_RESOURCE_GROUP="<resource group name>" # set to the name of the
resource group
export AZURE_LOCATION="westus"            # set to the location you are using
```

4. Set your KUBECONFIG environment variable:

```
export kubeconfig=${CLUSTER_NAME}.conf
```

5. Create the Secret with the Azure credentials, this will be used by the Azure CSI driver:
   a. Create an `azure.json` file:

```
cat <<EOF > azure.json
{
  "cloud": "AzurePublicCloud",
  "tenantId": "$AZURE_TENANT_ID",
  "subscriptionId": "$AZURE_SUBSCRIPTION_ID",
  "aadClientId": "$AZURE_CLIENT_ID",
  "aadClientSecret": "$AZURE_CLIENT_SECRET",
  "resourceGroup": "$AZURE_RESOURCE_GROUP",
  "location": "$AZURE_LOCATION"
}
EOF
```

   b. Create the Secret:

```
kubectl create secret generic azure-cloud-provider --namespace=kube-
system --type=Opaque --from-file=cloud-config=azure.json
```

6. Install the Azure Disk CSI driver:

```
$ curl -skSL https://raw.githubusercontent.com/kubernetes-sigs/azuredisk-csi-
driver/v1.26.2/deploy/install-driver.sh | bash -s v1.26.2 snapshot -
```

7. Check the status to see if the driver is ready for use:

```
kubectl -n kube-system get pod -o wide --watch -l app=csi-azuredisk-controller
kubectl -n kube-system get pod -o wide --watch -l app=csi-azuredisk-node
```

8. Now Kubernetes knows that this is Azure disk, and will create clusters on Azure. You are ready to create the StorageClass for the Azure Disk CSI Driver:

```
kubectl create -f https://raw.githubusercontent.com/kubernetes-sigs/azuredisk-csi-driver/master/deploy/example/storageclass-azuredisk-csi.yaml
```

9. Change the default storage class to this new StorageClass so that every new disk will be created in the Azure environment:

```
kubectl patch sc/localvolumeprovisioner -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class":"false"}}}'
kubectl patch sc/managed-csi -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

10. Verify that the StorageClass chosen is currently the default:

```
kubectl get storageclass
```

For more information about Azure Disk CSI for persistent storage and changing the default StorageClass, refer to that page in the documentation: Default Storage Providers in DKP (see page 110)

6.11.2.5.3 Next Step:

Pre-provisioned Modify the Calico Installation (see page 1095)

6.11.3 Pre-provisioned Cluster Creation Customization Choices

Below are a some methods to customize your cluster during creation. If none of these choices apply, skip this page and proceed to either section:

- Pre-provisioned Install in a Non-air-gapped Environment (see page 1105)
- Pre-provisioned Install in an Air-gapped Environment (see page 1120)

6.11.3.1 Section Topics

When creating clusters, many options are available such as those listed in this section of the documentation. A brief explanation of each choice is given in the following topic summaries with a link to the more descriptive page for that topic. To use these, proceed to the cluster choice page for detailed instructions:

- Pre-provisioned Customizing CAPI Clusters (see page 1085) — Familiarize yourself with Cluster API before editing the cluster objects because edits can prevent the cluster from deploying successfully.
- Pre-provisioned Registry Mirrors (see page 1085) — In an air-gapped environment, you need a local repository to store Helm charts, Docker images, and other artifacts. In an environment with access to the Internet, you can retrieve artifacts from specialized repositories dedicated to them, such as Docker images contained in DockerHub and Helm Charts that come from a dedicated Helm Chart repository.

- Pre-provisioned Create Secrets and Overrides (see page 1086) — Create necessary secrets and overrides for pre-provisioned clusters. Most applications deployed through Kubernetes https://kubernetes.io/docs/concepts/configuration/secret/ require access to databases, services, and other resources located externally. The easiest way to manage the login information necessary to access those resources is using secrets in order to help organize and distribute sensitive information across a cluster while minimizing the risk of sensitive information exposure.
- Pre-provisioned Define Control Plane Endpoint (see page 1092) — A control plane should have three, five, or seven nodes, so it can remain available if one, two, or three nodes fail. A control plane with one node should not be used in production.
- Pre-provisioned Configure MetalLB (see page 1093) — It is recommended that an external load balancer(LB) be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests. If you do not have one, you can use Metal LB to create a MetalLB configmap for your Pre-provisioned infrastructure.
- Pre-provisioned Modify the Calico Installation (see page 1095) — Calico is a networking and security solution that enables Kubernetes workloads and non-Kubernetes/legacy workloads to communicate seamlessly and securely. Sometimes changes are needed, so use the information in this Pre-provisioned Modify the Calico Installation page.
- Pre-provisioned Built-in Virtual IP (see page 1100) — As explained in Define the Control Plane Endpoint, we recommend using an external load balancer for the control plane endpoint, but provide a built-in virtual IP when an external load balancer is not available. If an external load balancer is not available, use the built-in virtual IP.
- Pre-provisioned Use HTTP Proxy (see page 1101) — When you require HTTP proxy configurations, you can apply them during the create operation by adding the appropriate flags to the dkp create cluster command.
- Pre-provisioned Use Alternate Pod or Service Subnets (see page 1102) — Some subnets are reserved by Kubernetes and can prevent proper cluster deployment if you unknowingly configure DKP so that the Node subnet collides with either the Pod or Service subnet.
- Pre-provisioned Output Directory YAML (see page 1104) — You can create individual files with different smaller manifests for ease in editing using the --output-directory flag used with --output=json|yaml. You create the directory of where to output resources to files.

6.11.3.1.1 Single-Node control plane

Do not use a single-node control plane in a production cluster.

A control plane with one node can use its single node as the endpoint, so you will not require an external load balancer, or a built-in virtual IP. At least one control plane node must always be running. Therefore, to upgrade a cluster with one control plane node, a spare machine must be available in the control plane inventory. This machine is used to provision the new node before the old node is deleted.

Modify Control Plane Audit logs settings using the information contained in the page Configuring the Control Plane (see page 1613).

When the API server endpoints are defined, you can create the cluster (see page 1107).

6.11.3.1.2 Next Step

If none of the customizations above applies, continue to installation instructions for your environment:

- Pre-provisioned Install in a Non-air-gapped Environment (see page 1105)
  OR
- Pre-provisioned Install in an Air-gapped Environment (see page 1120)

6.11.3.2 Pre-provisioned Customizing CAPI Clusters

Familiarize yourself with Cluster API (see page 1061) before editing the cluster objects because edits can prevent the cluster from deploying successfully.

The result of this command will allow such edits:

```
dkp create cluster preprovisioned \
    --cluster-name=${CLUSTER_NAME} \
    --dry-run \
    --output=yaml \
  > ${CLUSTER_NAME}.yaml
```

To edit the YAML, you need to understand the CAPI components to avoid breaking the cluster. For expanded component explanation, refer to the Universal Configurations for All Providers section of the documentation in the section: Customizing CAPI Components for a Cluster (see page 1061)

6.11.3.2.1 Next Topic

Pre-provisioned Registry Mirrors (see page 1085)

6.11.3.3 Pre-provisioned Registry Mirrors

In an air-gapped environment, you need a local repository to store Helm charts, Docker images, and other artifacts. In an environment with access to the Internet, you can retrieve artifacts from specialized repositories dedicated to them, such as Docker images contained in DockerHub and Helm Charts that come from a dedicated Helm Chart repository.

Kubernetes does not natively provide a registry for hosting the container images you will use to run the applications you want to deploy on Kubernetes. Instead, Kubernetes requires you to use an external solution for storing and sharing container images. There are a variety of Kubernetes-compatible registry options that are compatible with DKP.

6.11.3.3.1 How Does it Work?

The **first time** you request an image from your local registry mirror, it pulls the image from the public registry (such as Docker) and stores it locally before handing it back to you. On subsequent requests, the local registry mirror is able to serve the image from its own storage.

6.11.3.3.2 Air-gapped vs Non-air-gapped Environments

In a non-air-gapped environment, you have access to the Internet. You retrieve artifacts from specialized repositories dedicated to them, such as Docker images contained in DockerHub and Helm Charts that come from a dedicated Helm Chart repository. You can also create your own local repository to hold the downloaded container images needed or any custom images you've created with the Konvoy Image Builder (see page 1626) tool.

In an **air-gapped environment**, you need a local repository to store Helm charts, Docker images, and other artifacts. Private registries provide security and privacy into enterprise container image storage, whether hosted remotely or on-premises locally in an air-gapped environment. DKP in an air-gapped environment **requires** a local container registry of trusted images to enable production-level Kubernetes cluster management. However, a local registry is an option in a non-air-gapped environment as well for speed and security.

If you want to use images from this local registry to deploy applications inside your Kubernetes cluster, you'll need to set up a **secret** for a private registry. The secret contains your login data, which Kubernetes needs to connect to your private repository.

6.11.3.3.2.1 Related Topics

More information and detail can be found:

- Registry Mirror Tools (see page 1606)
- Use a Registry Mirror (see page 1609)

6.11.3.3.2.2 Next Topic

Pre-provisioned Create Secrets and Overrides (see page 1086)

6.11.3.4 Pre-provisioned Create Secrets and Overrides

Create necessary secrets and overrides for pre-provisioned clusters. Most applications deployed through Kubernetes[777] require access to databases, services, and other resources located externally. The easiest way to manage the login information necessary to access those resources is using secrets in order to help organize and distribute sensitive information across a cluster while minimizing the risk of sensitive information exposure.

DKP requires SSH access to your infrastructure with superuser privileges. You must provide an unencrypted SSH private key to DKP so secrets are a good way to achieve this. Populate the key and create the required secret, on your bootstrap cluster using the following procedure.

777 https://kubernetes.io/docs/concepts/configuration/secret/

6.11.3.4.1 Create a Unique Cluster Name

Give your cluster a unique name suitable for your environment.

Set the environment variable to be used throughout this procedure:

```
export CLUSTER_NAME=preprovisioned-example
```

(Optional) If you want to create a unique cluster name, use this command. This creates a unique name every time you run it, so use it carefully.

```
export CLUSTER_NAME=preprovisioned-example-$(LC_CTYPE=C tr -dc 'a-z0-9' </dev/urandom
| fold -w 5 | head -n1)
echo $CLUSTER_NAME
```

```
preprovisioned-example-pf4a3
```

6.11.3.4.2 Create a Secret

Create a secret that contains the SSH key with these commands:

```
export SSH_PRIVATE_KEY_FILE="<path-to-ssh-private-key>"
```

```
export SSH_PRIVATE_KEY_SECRET_NAME=$CLUSTER_NAME-ssh-key
```

```
kubectl create secret generic ${SSH_PRIVATE_KEY_SECRET_NAME} --from-file=ssh-
privatekey=${SSH_PRIVATE_KEY_FILE}
kubectl label secret ${SSH_PRIVATE_KEY_SECRET_NAME} clusterctl.cluster.x-k8s.io/move=
```

```
secret/preprovisioned-example-ssh-key created
secret/preprovisioned-example-ssh-key labeled
```

6.11.3.4.3 Create Overrides

In these steps, you will point your machines at the desired Registry to obtain the container images. If your pre-provisioned machines need to have Custom Override Files (see page 1677), create a secret that includes all the overrides you want to provide in one file.

1. Example CentOS7 and Docker:
   If you want to provide an override with Docker credentials and a different source for EPEL on a CentOS7 machine, you should create a file like this:

```
cat > overrides.yaml << EOF
image_registries_with_auth:
- host: "registry-1.docker.io"
  username: "my-user"
  password: "my-password"
  auth: ""
  identityToken: ""

epel_centos_7_rpm: https://my-rpm-repostory.org/epel/epel-release-
latest-7.noarch.rpm
EOF
```

   You can then create the related secret by running the following command:

```
kubectl create secret generic $CLUSTER_NAME-user-overrides --from-
file=overrides.yaml=overrides.yaml
kubectl label secret $CLUSTER_NAME-user-overrides clusterctl.cluster.x-k8s.io/
move=
```

2. Example:
   When using Oracle 7 OS, you may wish to deploy the RHCK kernel instead of the default UEK kernel. To do so, add the following text to your `overrides.yaml`:

```
cat > overrides.yaml << EOF
---
oracle_kernel: RHCK
EOF
```

   You can then create the related secret by running the following command:

```
kubectl create secret generic $CLUSTER_NAME-user-overrides --from-
file=overrides.yaml=overrides.yaml
kubectl label secret $CLUSTER_NAME-user-overrides clusterctl.cluster.x-k8s.io/
move=
```

6.11.3.4.3.1 Related Topic

Pre-provisioned FIPS Create Secrets and Overrides (see page 1089)

6.11.3.4.3.2 Next Topic

Pre-provisioned Define Control Plane Endpoint (see page 1092)

If none of the customizations apply, continue to installation instructions for your environment:

- Pre-provisioned Install in a Non-air-gapped Environment (see page 1105)
  OR
- Pre-provisioned Install in an Air-gapped Environment (see page 1120)

6.11.3.4.4 Pre-provisioned FIPS Create Secrets and Overrides

DKP requires SSH access to your infrastructure with superuser privileges. You must provide an unencrypted SSH private key to DKP so secrets are a good way to achieve this. Populate the key and create the required secret, on your bootstrap cluster using the following procedure.

6.11.3.4.4.1 Create a Unique Cluster Name

Give your cluster a unique name suitable for your environment.

Set the environment variable to be used throughout this procedure:

```
export CLUSTER_NAME=preprovisioned-example
```

(Optional) If you want to create a unique cluster name, use this command. This creates a unique name every time you run it, so use it carefully.

```
export CLUSTER_NAME=preprovisioned-example-$(LC_CTYPE=C tr -dc 'a-z0-9' </dev/urandom
| fold -w 5 | head -n1)
echo $CLUSTER_NAME
```

```
preprovisioned-example-pf4a3
```

6.11.3.4.4.2 Create a Secret

Create a secret that contains the SSH key with these commands:

```
export SSH_PRIVATE_KEY_FILE="<path-to-ssh-private-key>"
```

```
export SSH_PRIVATE_KEY_SECRET_NAME=$CLUSTER_NAME-ssh-key
```

```
kubectl create secret generic ${SSH_PRIVATE_KEY_SECRET_NAME} --from-file=ssh-
privatekey=${SSH_PRIVATE_KEY_FILE}
kubectl label secret ${SSH_PRIVATE_KEY_SECRET_NAME} clusterctl.cluster.x-k8s.io/move=
```

```
secret/preprovisioned-example-ssh-key created
secret/preprovisioned-example-ssh-key labeled
```

Non-air-gapped Environment Create FIPS-140 images

KIB can produce images containing FIPS-140 compliant binaries. Use the `fips.yaml` override file (see page 1671) provided with the image bundles.

You can also find these override files in the Konvoy Image Builder repo[778].

6.11.3.4.4.3 Create Overrides

1. Create a secret that includes the customization Overrides for FIPS compliance:
   Note: Get the latest values for FIPS from the Konvoy Image Builder repo[779].

   ```
   cat > overrides.yaml << EOF
   ---
   k8s_image_registry: docker.io/mesosphere

   fips:
     enabled: true

   build_name_extra: -fips
   kubernetes_build_metadata: fips.0
   default_image_repo: hub.docker.io/mesosphere
   kubernetes_rpm_repository_url: "https://packages.d2iq.com/konvoy/stable/linux/
   repos/el/kubernetes-v{{ kubernetes_version }}-fips/x86_64"
   docker_rpm_repository_url: "\
     https://containerd-fips.s3.us-east-2.amazonaws.com\
     /{{ ansible_distribution_major_version|int }}\
     /x86_64"
   EOF
   ```

778 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides
779 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

2. If your pre-provisioned machines need to have a customization with alternate package libraries, Docker image or other container registry image repos, or other Custom Override Files (see page 1677), **add** more lines to the same Overrides file.
    a. Example:
    If you want to provide an override with Docker credentials and a different source for EPEL on a CentOS7 machine, you should create a file like this:

```
cat > overrides.yaml << EOF
---
# fips configuration
k8s_image_registry: docker.io/mesosphere

fips:
  enabled: true

build_name_extra: -fips
kubernetes_build_metadata: fips.0
default_image_repo: hub.docker.io/mesosphere
kubernetes_rpm_repository_url: "https://packages.d2iq.com/konvoy/stable/
linux/repos/el/kubernetes-v{{ kubernetes_version }}-fips/x86_64"
docker_rpm_repository_url: "\
  https://containerd-fips.s3.us-east-2.amazonaws.com\
  /{{ ansible_distribution_major_version|int }}\
  /x86_64"

# custom configuration
image_registries_with_auth:
- host: "registry-1.docker.io"
  username: "my-user"
  password: "my-password"
  auth: ""
  identityToken: ""

epel_centos_7_rpm: https://my-rpm-repostory.org/epel/epel-release-
latest-7.noarch.rpm
EOF
```

    b. Example:
    When using Oracle 7 OS, you may wish to deploy the RHCK kernel instead of the default UEK kernel. To do so, add the following text to your `overrides.yaml`:

```
cat > overrides.yaml << EOF
---
# fips configuration
k8s_image_registry: docker.io/mesosphere
```

```
fips:
  enabled: true

build_name_extra: -fips
kubernetes_build_metadata: fips.0
default_image_repo: hub.docker.io/mesosphere
kubernetes_rpm_repository_url: "https://packages.d2iq.com/konvoy/stable/
linux/repos/el/kubernetes-v{{ kubernetes_version }}-fips/x86_64"
docker_rpm_repository_url: "\
  https://containerd-fips.s3.us-east-2.amazonaws.com\
  /{{ ansible_distribution_major_version|int }}\
  /x86_64"

# custom configuration
oracle_kernel: RHCK
EOF
```

3. Create the related secret by running the following command:

```
kubectl create secret generic $CLUSTER_NAME-user-overrides --from-
file=overrides.yaml=overrides.yaml
kubectl label secret $CLUSTER_NAME-user-overrides clusterctl.cluster.x-k8s.io/
move=
```

Next Step

Pre-provisioned Define Control Plane Endpoint (see page 1092)

If none of the customizations apply, continue to installation instructions for your environment:

- Pre-provisioned Install in a Non-air-gapped Environment (see page 1105)
  OR
- Pre-provisioned Install in an Air-gapped Environment (see page 1120)

6.11.3.5 Pre-provisioned Define Control Plane Endpoint

A control plane should have three, five, or seven nodes, so it can remain available if one, two, or three nodes fail. A control plane with one node should not be used in production.

In addition, the control plane should have an endpoint that remains available if some nodes fail.

```
                        -------- cp1.example.com:6443
                        |
lb.example.com:6443 ---------- cp2.example.com:6443
                        |
                        -------- cp3.example.com:6443
```

In this example, the control plane endpoint host is `lb.example.com`, and the control plane endpoint port is `6443`. The control plane nodes are `cp1.example.com`, `cp2.example.com`, and `cp3.example.com`. The port of each API server is `6443`.

The control plane endpoint port is also used as the API server port on each control plane machine. The default port is 6443. Before you create the cluster, ensure the port is available for use on each control plane machine.

6.11.3.5.1 Next Step

Pre-provisioned Configure MetalLB (see page 1093)

If none of the customizations apply, continue to installation instructions for your environment:

- Pre-provisioned Install in a Non-air-gapped Environment (see page 1105)
  OR
- Pre-provisioned Install in an Air-gapped Environment (see page 1120)

6.11.3.6 Pre-provisioned Configure MetalLB

It is recommended that an external load balancer(LB) be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests. If you do not have one, you can use Metal LB to create a MetalLB configmap for your Pre-provisioned infrastructure.

Choose one of the following two protocols you want to use to announce service IPs. If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can continue the installation process.

To use MetalLB, create a MetalLB configMap for your Pre-provisioned infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

6.11.3.6.1 Layer 2 Configuration

Layer 2 mode is the easiest to configure in many cases, because you do not need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

- MetalLB IP address ranges/CIDRs should be within the node's primary network subnet (see page 1065).

- MetalLB IP address ranges/CIDRs and node subnet should not conflict with the Kubernetes cluster pod and service subnets.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

After completion, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

6.11.3.6.2 BGP Configuration

For a basic configuration featuring one BGP router and one IP address range, you need 4 pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

After completion, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

6.11.3.6.2.1 Next Step

Pre-provisioned Modify the Calico Installation (see page 1095)

If none of the customizations apply, continue to installation instructions for your environment:

- Pre-provisioned Install in a Non-air-gapped Environment (see page 1105)
  OR
- Pre-provisioned Install in an Air-gapped Environment (see page 1120)

6.11.3.7 Pre-provisioned Modify the Calico Installation

Calico is a networking and security solution that enables Kubernetes workloads and non-Kubernetes/legacy workloads to communicate seamlessly and securely. Sometimes changes are needed, so use the information in this Pre-provisioned Modify the Calico Installation page.

6.11.3.7.1 Set the Interface

By default, Calico automatically detects the IP address to use for each node using the `first-found` method[780]. This is not always appropriate for your particular nodes. In that case, you must modify Calico's configuration to use a different method. An alternative is to use the `interface` method by providing the interface ID. Follow the steps outlined in this section to modify Calico's configuration.

> Azure does **not** set the interface. Proceed to Change the Encapsulation Type (see page 1098) section below.

In this example, all cluster nodes use `ens192` as the interface name.

1. Get the pods running on your cluster with this command:

```
kubectl get pods -A --kubeconfig ${CLUSTER_NAME}.conf
```

```
NAMESPACE                NAME
READY   STATUS            RESTARTS        AGE
calico-system            calico-kube-controllers-57fbd7bd59-vpn8b
1/1     Running           0               16m
calico-system            calico-node-5tbvl
1/1     Running           0               16m
calico-system            calico-node-nbdwd
1/1     Running           0               4m40s
calico-system            calico-node-twl6b
0/1     PodInitializing   0               9s
calico-system            calico-node-wktkh
1/1     Running           0               5m35s
calico-system            calico-typha-54f46b998d-52pt2
1/1     Running           0               16m
calico-system            calico-typha-54f46b998d-9tzb8
1/1     Running           0               4m31s
default                  cuda-vectoradd
0/1     Pending           0               0s
kube-system              coredns-78fcd69978-frwx4
1/1     Running           0               16m
kube-system              coredns-78fcd69978-kkf44
1/1     Running           0               16m
kube-system              etcd-ip-10-0-121-16.us-west-2.compute.internal
0/1     Running           0               8s
```

780 https://projectcalico.docs.tigera.io/reference/node/configuration#ip-autodetection-methods

```
kube-system              etcd-ip-10-0-46-17.us-west-2.compute.internal
1/1     Running           1               16m
kube-system              etcd-ip-10-0-88-238.us-west-2.compute.internal
1/1     Running           1               5m35s
kube-system              kube-apiserver-ip-10-0-121-16.us-west-2.compute.internal
0/1     Running           6               7s
kube-system              kube-apiserver-ip-10-0-46-17.us-west-2.compute.internal
1/1     Running           1               16m
kube-system              kube-apiserver-ip-10-0-88-238.us-west-2.compute.internal
1/1     Running           1               5m34s
kube-system              kube-controller-manager-ip-10-0-121-16.us-west-2.compute.int
ernal   0/1     Running          0               7s
kube-system              kube-controller-manager-ip-10-0-46-17.us-west-2.compute.inte
rnal    1/1     Running          1 (5m25s ago)   15m
kube-system              kube-controller-manager-ip-10-0-88-238.us-west-2.compute.int
ernal   1/1     Running          0               5m34s
kube-system              kube-proxy-gclmt
1/1     Running           0               16m
kube-system              kube-proxy-gptd4
1/1     Running           0               9s
kube-system              kube-proxy-mwkgl
1/1     Running           0               4m40s
kube-system              kube-proxy-zcqxd
1/1     Running           0               5m35s
kube-system              kube-scheduler-ip-10-0-121-16.us-west-2.compute.internal
0/1     Running           1               7s
kube-system              kube-scheduler-ip-10-0-46-17.us-west-2.compute.internal
1/1     Running           3 (5m25s ago)   16m
kube-system              kube-scheduler-ip-10-0-88-238.us-west-2.compute.internal
1/1     Running           1               5m34s
kube-system              local-volume-provisioner-2mv7z
1/1     Running           0               4m10s
kube-system              local-volume-provisioner-vdcrg
1/1     Running           0               4m53s
kube-system              local-volume-provisioner-wsjrt
1/1     Running           0               16m
node-feature-discovery   node-feature-discovery-master-84c67dcbb6-m78vr
1/1     Running           0               16m
node-feature-discovery   node-feature-discovery-worker-vpvpl
1/1     Running           0               4m10s
tigera-operator          tigera-operator-d499f5c8f-79dc4
1/1     Running           1 (5m24s ago)   16m
```

If a `calico-node` pod is not ready on your cluster, you must edit the `default` Installation resource. To edit the Installation resource, run the command:

```
kubectl edit installation default --kubeconfig ${CLUSTER_NAME}.conf
```

2. Change the value for `spec.calicoNetwork.nodeAddressAutodetectionV4` to `interface: ens192`, and save the resource:

```
spec:
  calicoNetwork:
  ...
    nodeAddressAutodetectionV4:
      interface: ens192
```

3. Save this resource. You may need to delete the node feature discovery worker pod in the `node-feature-discovery` namespace if that pod has failed. After you delete it, Kubernetes replaces the pod as part of its normal reconciliation.

6.11.3.7.1.1 Change the Encapsulation Type

Calico can leverage different network encapsulation methods to route traffic for your workloads. Encapsulation is useful when running on top of an underlying network that is not aware of workload IPs. Common examples of this include:

- Public cloud environments where you don't own the hardware.
- AWS across VPC subnet boundaries.
- Environments where you cannot peer Calico over BGP to the underlay or easily configure static routes.

WARNING: Switching encapsulation modes can cause disruption to in-progress connections. You can do this safely when the cluster is first deployed. However, if user workloads are already running on the cluster, plan accordingly for interruption.

6.11.3.7.1.2 Provider Specific Settings

The encapsulation type to be used for networking depends on the cloud provider. IP-in-IP is the default encapsulation method for Calico which most providers use, but not Azure.

Azure only supports **VXLAN** encapsulation type. Therefore, if you install on Azure pre-provisioned VMs, you must set the encapsulation mode to VXLAN.

D2iQ recommends changing the encapsulation type after cluster creation but before production using the method below. To change the encapsulation type, follow these steps:

1. First, remove the existing `default-ipv4-ippool` IPPool resource from `kubeconfig`. The resource must be deleted, so that it can re-create after you edit the Installation resource. Execute the command below to delete:

```
kubectl delete ippool default-ipv4-ippool
```

2. Run the following command to edit:

```
kubectl edit installation default --kubeconfig ${CLUSTER_NAME}.conf
```

3. Change the value for encapsulation `- encapsulation:` as shown below:

```
  spec:
  calicoNetwork:
    ipPools:
      - encapsulation: VXLAN
```

VXLAN

VXLAN is a tunneling protocol that encapsulates layer 2 Ethernet frames in UDP packets, enabling you to create virtualized layer 2 subnets that span Layer 3 networks. It has a slightly larger header than IP-in-IP which creates a slight reduction in performance over IP-in-IP.

IPIP

IP-in-IP is an IP tunneling protocol that encapsulates one IP packet in another IP packet. An outer packet header is added with the tunnel entry point and the tunnel exit point. The calico implementation of this protocol uses BGP to determine the exit point making this protocol unusable on networks that don't pass BGP.

For more information, see:

- Calico Overlay Networking[781]
- Calico Routing for VXLAN[782]
- IP-in-IP RFC 2003[783]
- VXLAN RFC 7348[784]

781 https://docs.projectcalico.org/networking/vxlan-ipip
782 https://joshrosso.com/docs/2020/2020-10-01-calico-routing-modes/
783 https://datatracker.ietf.org/doc/html/rfc2003
784 https://datatracker.ietf.org/doc/html/rfc7348

If using Windows, see this documentation on Calico site regarding limitations: Calico for Windows VXLAN[785]

6.11.3.7.1.3 Next Step

Pre-provisioned Built-in Virtual IP (see page 1100)

If none of the customizations apply, continue to installation instructions for your environment:

- Pre-provisioned Install in a Non-air-gapped Environment (see page 1105)
  OR
- Pre-provisioned Install in an Air-gapped Environment (see page 1120)

6.11.3.8 Pre-provisioned Built-in Virtual IP

As explained in Define the Control Plane Endpoint (see page 151), we recommend using an external load balancer for the control plane endpoint, but provide a built-in virtual IP when an external load balancer is not available. If an external load balancer is not available, use the built-in virtual IP.

The virtual IP is *not* a load balancer; it does not distribute request load among the control plane machines. However, if the machine receiving requests does not respond to them, the virtual IP automatically moves to another machine. The built-in virtual IP uses the kube-vip[786] project. To use the virtual IP, add these flags to the `create cluster` command:

Virtual IP Configuration	Flag
Network interface to use for Virtual IP. *Must exist on all control plane machines.*	`--virtual-ip-interface string`
IPv4 address. *Reserved for use by the cluster.*	`--control-plane-endpoint string`

6.11.3.8.1 Virtual IP Example

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --control-plane-endpoint-host 196.168.1.10 \
    --virtual-ip-interface eth1
    --pre-provisioned-inventory-file preprovisioned_inventory.yaml
```

785 https://projectcalico.docs.tigera.io/getting-started/windows-calico/limitations#calico-vxlan-networking-limitations
786 https://kube-vip.io/

```
    --ssh-private-key-file <path-to-ssh-private-key>
    --self-managed
```

6.11.3.8.2 Next Step

Pre-provisioned Use HTTP Proxy (see page 1101)

If none of the customizations apply, continue to installation instructions for your environment:

- Pre-provisioned Install in a Non-air-gapped Environment (see page 1105)
  OR
- Pre-provisioned Install in an Air-gapped Environment (see page 1120)

6.11.3.9 Pre-provisioned Use HTTP Proxy

When you require HTTP proxy configurations, you can apply them during the `create` operation by adding the appropriate flags to the `dkp create cluster` command.

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. Prior to DKP 2.6, you had to specify the HTTP proxy in the KIB override setup and then again in the `dkp create cluster` command. After DKP 2.6, an HTTP proxy gets created from the Konvoy flags for the control plane proxy and workers proxy values. The flags in the DKP command for Pre-provisioned clusters populate a Secret automatically in the bootstrap cluster. That Secret has a known name that the Pre-provisioned controller finds and applies when it runs the KIB provisioning job.

More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

Proxy configuration	Flag
HTTP proxy for control plane machines	`--control-plane-http-proxy string`
HTTPS proxy for control plane machines	`--control-plane-https-proxy string`
No Proxy list for control plane machines	`--control-plane-no-proxy strings`
HTTP proxy for worker machines	`--worker-http-proxy string`
HTTPS proxy for worker machines	`--worker-https-proxy string`
No Proxy list for worker machines	`--worker-no-proxy strings`

You must also add the same configuration as an override (see page 1682). For more information, refer to this documentation (see page 1677).

6.11.3.9.1 HTTP Proxy Example

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --control-plane-http-proxy http://proxy.example.com:8080 \
    --control-plane-https-proxy https://proxy.example.com:8080 \
    --control-plane-no-proxy
"127.0.0.1,10.96.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default.svc,kubernetes.d
efault.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.cluste
r.local" \
    --worker-http-proxy http://proxy.example.com:8080 \
    --worker-https-proxy https://proxy.example.com:8080 \
    --worker-no-proxy
"127.0.0.1,10.96.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default.svc,kubernetes.d
efault.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.cluste
r.local"
```

6.11.3.9.2 Related Topic

Pre-Provisioned Override Files (see page 1680)

6.11.3.9.3 Next Step

Pre-provisioned Use Alternate Pod or Service Subnets (see page 1102)

If none of the customizations apply, continue to installation instructions for your environment:

- Pre-provisioned Install in a Non-air-gapped Environment (see page 1105)
  OR
- Pre-provisioned Install in an Air-gapped Environment (see page 1120)

6.11.3.10 Pre-provisioned Use Alternate Pod or Service Subnets

Some subnets are reserved by Kubernetes and can prevent proper cluster deployment if you unknowingly configure DKP so that the Node subnet collides with either the Pod or Service subnet.

In Konvoy, the default pod subnet is 192.168.0.0/16, and the default service subnet is 10.96.0.0/12. Ensure your subnets do not overlap with your host subnet because they cannot be changed after cluster creation.

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

If you need to change the Kubernetes subnets, you must do this at cluster creation. To change the subnets, perform the following steps:

1. Generate the YAML manifests for the cluster using the `--dry-run` and `-o yaml` flags, along with the desired `dkp cluster create` command:

   ```
   dkp create cluster preprovisioned --cluster-name ${CLUSTER_NAME} --control-
   plane-endpoint-host <control plane endpoint host> --control-plane-endpoint-port
   <control plane endpoint port, if different than 6443> --dry-run -o yaml >
   cluster.yaml
   ```

2. To modify the service subnet, add or edit the `spec.clusterNetwork.services.cidrBlocks` field of the `Cluster` object:

   ```
   kind: Cluster
   spec:
     clusterNetwork:
       services:
         cidrBlocks:
         - 10.0.0.0/12
   ```

3. To modify the pod subnet, edit the `Cluster` and calico-cni `ConfigMap` resources:

   Cluster: Add or edit the `spec.clusterNetwork.pods.cidrBlocks` field:

   ```
   kind: Cluster
   spec:
     clusterNetwork:
       pods:
         cidrBlocks:
         - 172.16.0.0/16
   ```

   ConfigMap: Edit the `data."custom-resources.yaml".spec.calicoNetwork.ipPools.cidr` field with your desired pod subnet:

```
apiVersion: v1
data:
  custom-resources.yaml: |
    apiVersion: operator.tigera.io/v1
    kind: Installation
    metadata:
      name: default
    spec:
      # Configures Calico networking.
      calicoNetwork:
        # Note: The ipPools section cannot be modified post-install.
        ipPools:
        - blockSize: 26
          cidr: 172.16.0.0/16
kind: ConfigMap
metadata:
  name: calico-cni-<cluter-name>
```

When you provision the cluster, the configured pod and service subnets will be applied.

6.11.3.10.1 Next Step

Pre-provisioned Output Directory YAML (see page 1104)

If none of the customizations apply, continue to installation instructions for your environment:

- Pre-provisioned Install in a Non-air-gapped Environment (see page 1105)
  OR
- Pre-provisioned Install in an Air-gapped Environment (see page 1120)

6.11.3.11 Pre-provisioned Output Directory YAML

You can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag used with `--output=json|yaml`. You create the directory of where to output resources to files.

Using this flag will create multiple files in the specified directory which must already exist:

```
dkp create cluster preprovisioned
    --cluster-name=${CLUSTER_NAME} \
    --dry-run \
    --output=yaml \
    --output-directory=<existing-directory>
```

For more information regarding this flag or others, please refer to the CLI section of the documentation for the dkp create cluster (see page 1853) command and select your provider.

6.11.3.11.1 Next Step

- Pre-provisioned Install in a Non-air-gapped Environment (see page 1105)
  OR
- Pre-provisioned Install in an Air-gapped Environment (see page 1120)

6.11.4 Pre-provisioned Install in a Non-air-gapped Environment

In pre-provisioned environments, DKP handles your cluster's lifecycle including installation, upgrade, and node management. DKP installs Kubernetes, monitoring and logging apps, as well as its own user interface.

In an environment with access to the Internet, you retrieve artifacts from specialized repositories dedicated to them, such as Docker® images contained in DockerHub, and Helm™ Charts that come from a dedicated Helm Chart repository. However, in an air-gapped environment, you need local repositories to store Helm charts, Docker images, and other artifacts. Tools such as JFrog™, Harbor™, and Nexus™ handle multiple types of artifacts in one local repository.

If desired, a local registry can also be used in a non-air-gapped environment for purposes of speed and security. To do so, add the Pre-provisioned Air-gapped Define Environment (see page 1120) steps to your non-air-gapped installation process.

6.11.4.1 Installation Process

- Pre-provisioned Bootstrap Cluster (see page 1105)
- Pre-provisioned Create a New Cluster (see page 1107)
- Pre-provisioned Make your Cluster Self-managed (see page 1113)
- Pre-provisioned Install Kommander Non-air-gapped (see page 1116)

6.11.4.2 Pre-provisioned Bootstrap Cluster

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster. The workload cluster then manages its own lifecycle.

To create Kubernetes clusters, Konvoy uses Cluster API[787] (CAPI) controllers. These controllers run on a Kubernetes cluster. To get started, you need a *bootstrap* cluster. By default, Konvoy creates a bootstrap cluster for you in a Docker container using the Kubernetes-in-Docker (KIND[788]) tool.

6.11.4.2.1 Prerequisites

Before you begin, you must:

- Complete the steps in Prerequisites (see page 141).
- Ensure the `dkp` binary can be found in your $PATH.
- IF using a Registry Mirror even though you are not in an air-gapped environment, refer to the air-gapped section for loading images: Pre-provisioned Air-gapped Define Environment (see page 1120)

6.11.4.2.1.1 Bootstrap Cluster Lifecycle Services

1. Review Universal Configurations for all Infrastructure Providers (see page 1052) regarding settings, flags and other choices and then begin bootstrapping.
2. Create a bootstrap cluster:

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config
   ```

   - Configuring an HTTP/HTTPS Proxy (see page 1053) use `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful.

HTTP Flags if needed:

To create a bootstrap cluster in a proxied environment (see page 1054) use this command syntax, in addition to any other flags you may need:

```
--http-proxy <string> \
--https-proxy <string> \
--no-proxy <string>
```

Output:

```
✓ Creating a bootstrap cluster
✓ Initializing new CAPI components
```

Konvoy creates a bootstrap cluster using KIND[789] as a library and deploys Cluster API[790] providers on the cluster. Refer to Customizing CAPI Components for a Cluster (see page 1061)for more details.

787 https://cluster-api.sigs.k8s.io/
788 https://github.com/kubernetes-sigs/kind
789 https://github.com/kubernetes-sigs/kind
790 https://cluster-api.sigs.k8s.io/

Konvoy waits until the controller-manager and webhook deployments of these providers are ready. List these deployments using this command:

```
kubectl get --all-namespaces deployments -l=clusterctl.cluster.x-k8s.io
```

Output:

```
NAMESPACE                           NAME
READY   UP-TO-DATE   AVAILABLE   AGE
capa-system                         capa-controller-manager                         1
/1      1            1           2m8s
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-manager       1
/1      1            1           2m10s
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-manager   1
/1      1            1           2m10s
capi-system                         capi-controller-manager                         1
/1      1            1           2m11s
cappp-system                        cappp-controller-manager                        1
/1      1            1           2m6s
capv-system                         capv-controller-manager                         1
/1      1            1           2m5s
capz-system                         capz-controller-manager                         1
/1      1            1           2m7s
cert-manager                        cert-manager                                    1
/1      1            1           2m21s
cert-manager                        cert-manager-cainjector                         1
/1      1            1           2m21s
cert-manager                        cert-manager-webhook                            1
/1      1            1           2m21s
```

6.11.4.2.1.2 Next Step

Pre-provisioned Create a New Cluster (see page 1107)

6.11.4.3 Pre-provisioned Create a New Cluster

After you have defined the infrastructure (see page 1074) and control plane endpoints (see page 1092), you create a Kubernetes cluster using the infrastructure definition. Create the cluster by following these steps to spin up a new pre-provisioned cluster.

Before you create a new DKP cluster below, you may choose an external load balancer or virtual IP and use the corresponding `dkp create cluster` command example from that page in the docs from the links below. Other customizations are available, but require different flags during `dkp create cluster` command also. Refer to Pre-provisioned Cluster Creation Customization Choices (see page 1083) for more cluster customizations.

DKP uses `localvolumeprovisioner` as the default storage provider (see page 110). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[791] compatible storage that is suitable for production.

You can choose from any of the storage options[792] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[793].

NOTE: When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.

NOTE: To increase Docker Hub's rate limit[794] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.

NOTE: Ensure your subnets (see page 1102) do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

Generate the Kubernetes cluster objects:

791 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
792 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
793 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/
794 https://docs.docker.com/docker-hub/download-rate-limit/

- The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory. It uses the default external load balancer.
    a. **(Optional)** If you have overrides for your clusters[795], you must specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied.
    `--override-secret-name=$CLUSTER_NAME-user-overrides`
    b. **(Optional)** Use a registry mirror (see page 1609). Configure your cluster to use an existing local registry (see page 1606) as a mirror when attempting to pull images previously pushed to your registry (see page 1120). Instructions in the expandable section:

Export Registry Variables and Flags for Cluster Creation:

If you have a local registry, you must provide additional arguments when creating the cluster. These tell the cluster where to locate the local registry to use by defining the URL. Set the needed environment variable(s) with your registry information:

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_CA=<path to the CA on the bastion>
export REGISTRY_USERNAME=<username>
export REGISTRY_USERNAME=<password>
```

- `REGISTRY_URL` : the address of an existing container registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- `REGISTRY_CA` : (optional) the path on the bastion machine to the container registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.

When **creating the cluster**, apply the variables you defined above during the `dkp create cluster` command with the flags needed for your environment:

```
--registry-mirror-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD}
```

1. **Create cluster command** - Depending on the cluster size, it will take a few minutes to create the Kubernetes cluster objects:

```
dkp create cluster preprovisioned \
  --cluster-name ${CLUSTER_NAME} \
```

795 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29918736

```
  --control-plane-endpoint-host <control plane endpoint host> \
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443> \
  --override-secret-name=$CLUSTER_NAME-user-overrides \
  --dry-run \
  --output=yaml \
  > ${CLUSTER_NAME}.yaml
```

a. **Virtual IP ALTERNATIVE** - if you don't have an external LB, and wish to use a VIRTUAL IP provided by kube-vip, specify these flags example below:

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --control-plane-endpoint-host 196.168.1.10 \
    --virtual-ip-interface eth1 \
    --dry-run \
    --output=yaml \
    > ${CLUSTER_NAME}.yaml
```

HTTP only:

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

```
--http-proxy <<http proxy list>>
--https-proxy <<https proxy list>>
--no-proxy <<no proxy list>>
```

- To configure the Control Plane and Worker nodes to use an HTTP proxy:

```
export CONTROL_PLANE_HTTP_PROXY=http://example.org:8080
export CONTROL_PLANE_HTTPS_PROXY=http://example.org:8080
export
CONTROL_PLANE_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1
,10.96.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.s
vc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.sv
c.cluster,.svc.cluster.local,169.254.169.254"

export WORKER_HTTP_PROXY=http://example.org:8080
export WORKER_HTTPS_PROXY=http://example.org:8080
export
WORKER_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96.
0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kube
rnetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.clust
er,.svc.cluster.local,169.254.169.254"
```

- Replace:
  - `example.org,example.com,example.net` with you internal addresses
  - `localhost` and `127.0.0.1` addresses should not use the proxy
  - `10.96.0.0/12` is the default Kubernetes service subnet
  - `192.168.0.0/16` is the default Kubernetes pod subnet
  - `kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local` is the internal Kubernetes kube-apiserver service
  - `.svc,.svc.cluster,.svc.cluster.local` is the internal Kubernetes services
- Use the flags in the `dkp create cluster` command:

```
--control-plane-http-proxy="${CONTROL_PLANE_HTTP_PROXY}" \
--control-plane-https-proxy="${CONTROL_PLANE_HTTPS_PROXY}" \
--control-plane-no-proxy="${CONTROL_PLANE_NO_PROXY}" \
--worker-http-proxy="${WORKER_HTTP_PROXY}" \
--worker-https-proxy="${WORKER_HTTPS_PROXY}" \
--worker-no-proxy="${WORKER_NO_PROXY}" \
```

FIPS Requirements

To create a cluster in FIPS mode (see page 1598), inform the controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes by adding those flags to `dkp create cluster` command:

```
--kubernetes-version=v1.28.7+fips.0 \
--etcd-version=3.5.10+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
```

Individual manifests using the Output Directory flag:

You can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag. This will create multiple files in the specified directory which must already exist:

```
--output-directory=<existing-directory>
```

Refer to the Cluster Creation Customization Choices (see page 1083) section for more information on how to use optional flags such as the `--output-directory` flag.

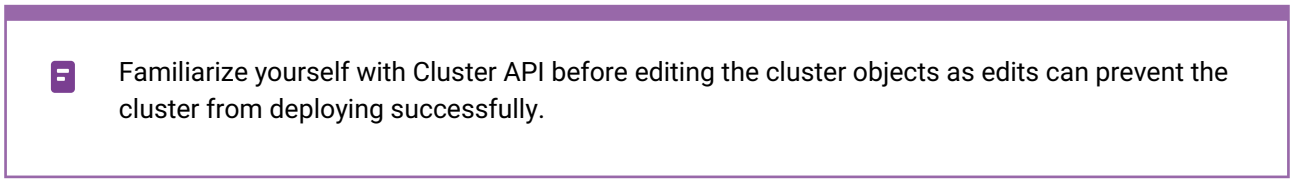
Cluster creation results:

Output

The output from this command is shortened here for reading clarity, but should start like this:

```
Generating cluster resources
cluster.cluster.x-k8s.io/preprovisioned-example created
cont.........
```

2. Inspect or edit the cluster objects and familiarize yourself with Cluster API before editing the cluster objects as edits can prevent the cluster from deploying successfully. See Pre-provisioned Customizing CAPI Clusters (see page 1085) .



3. Create the cluster from the objects generated in the `dry run`. A warning will appear in the console if the resource already exists and will require you to remove the resource or update your YAML.

   ```
   kubectl create -f ${CLUSTER_NAME}.yaml
   ```

   NOTE: If you used the `--output-directory` flag in your `dkp create .. --dry-run` step above, create the cluster from the objects you created by specifying the directory:

   ```
   kubectl create -f <existing-directory>/
   ```

4. Use the wait command to monitor the cluster control-plane readiness:

   ```
   kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
   timeout=30m
   ```

   Output:

   ```
   cluster.cluster.x-k8s.io/preprovisioned-example condition met
   ```

5. After the creation, use this command to get the Kubernetes kubeconfig for the new cluster and begin deploying workloads:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

Azure requires changing the CNI encapsulation type of Calico from the default of IPtoIP to VXlan. If changing the Calico encapsulation (see page 1095), D2iQ recommends changing it after cluster creation, but before production.

6.11.4.3.1 Audit logs

To modify Control Plane Audit logs settings using the information contained in the page Configuring the Control Plane (see page 1613).

6.11.4.3.1.1 Further Optional Steps

- Azure Only Configurations (see page 1079)
- Modify the Calico installation (see page 1095)

6.11.4.3.1.2 Next Step

When you complete this procedure, move on to Pre-provisioned Make your Cluster Self-managed (see page 1113) to continue the process.

6.11.4.4 Pre-provisioned Make your Cluster Self-managed

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which then deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, which is now self-managed. This guide describes how to make a workload cluster self-managed.

This page contains instructions on how to make your cluster self-managed. This is necessary if there is only one cluster in your environment, or if this cluster should become the Management cluster in a multi-cluster environment.

If you already have a self-managed or Management cluster in your environment, skip this page.

6.11.4.4.1 Make the New Kubernetes Cluster Manage Itself

Your new cluster is turned into a **Management Cluster** (or free standing Essential Cluster) using this procedure:

If you have not already retrieved the kubeconfig after creating the cluster, use this command before proceeding: `dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf`

1. Deploy cluster lifecycle services on the workload cluster:

```
dkp create capi-components --kubeconfig ${CLUSTER_NAME}.conf
```

Output:

```
✓ Initializing new CAPI components
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

2. Move the Cluster API objects from the bootstrap to the workload cluster:

The cluster lifecycle services on the workload cluster are ready, but the workload cluster configuration is on the bootstrap cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the bootstrap to the workload cluster. This process is also called a Pivot[796]. First unset the kubeconfig and then move the CAPI:

```
unset KUBECONFIG
```

Next:

```
dkp move capi-resources --to-kubeconfig ${CLUSTER_NAME}.conf
```

Output:

```
✓ Moving cluster resources
You can now view resources in the moved cluster by using the --kubeconfig flag
with kubectl. For example: kubectl --kubeconfig=preprovisioned-example.conf get
nodes
```

796 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

NOTE: To ensure only one set of cluster lifecycle services manages the workload cluster, Konvoy first pauses reconciliation of the objects on the bootstrap cluster, then creates the objects on the workload cluster. As Konvoy copies the objects, the cluster lifecycle services on the workload cluster reconcile the objects. The workload cluster becomes self-managed after Konvoy creates all the objects. If it fails, the `move` command can be safely retried.

3. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf wait --for=condition=ControlPlaneRead
y "clusters/${CLUSTER_NAME}" --timeout=20m
```

Output:

```
cluster.cluster.x-k8s.io preprovisioned-example condition met
```

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use Konvoy with the workload cluster kubeconfig.

```
dkp describe cluster --kubeconfig ${CLUSTER_NAME}.conf -c ${CLUSTER_NAME}
```

Output:

```
NAME
READY  SEVERITY  REASON  SINCE  MESSAGE
Cluster/preprovisioned-example                                          True
2m31s
├─ClusterInfrastructure - PreprovisionedCluster/preprovisioned-example
├─ControlPlane - KubeadmControlPlane/preprovisioned-example-control-plane  True
2m31s
│ ├─Machine/preprovisioned-example-control-plane-6g6nr                  True
2m33s
│ ├─Machine/preprovisioned-example-control-plane-8lhcv                  True
2m33s
│ └─Machine/preprovisioned-example-control-plane-kk2kg                  True
2m33s
└─Workers
  └─MachineDeployment/preprovisioned-example-md-0                       True
2m34s
    └─Machine/preprovisioned-example-md-0-77f667cd9-tnctd               True
2m33s
```

5. Remove the bootstrap cluster, as the workload cluster is now self-managed:

```
dkp delete bootstrap
```

Output:

```
✓ Deleting bootstrap cluster
```

6.11.4.4.1.1 Known Limitations

Be aware of these limitations in the current release of Konvoy.

- DKP supports moving only one set of cluster objects from the bootstrap cluster to the workload cluster, or vice-versa.
- DKP only supports moving all namespaces in the cluster; DKP does not support migration of individual namespaces.

6.11.4.4.1.2 Next Step

Pre-provisioned Install Kommander Non-air-gapped (see page 1116)

6.11.4.5 Pre-provisioned Install Kommander Non-air-gapped

This section provides installation instructions for the Kommander component of DKP in a non-air-gapped pre-provisioned environment.

6.11.4.5.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

6.11.4.5.1.1 Create and Customize your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1558) for the deployment:

```
dkp install kommander --init > kommander.yaml
```

4. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP's default configuration ships Ceph with PVC based storage[797] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner (see page 110), you can install Ceph in host storage[798] mode.

   You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

   a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

```
...
rook-ceph-cluster:
    enabled: true
    values: |
      cephClusterSpec:
        storage:
          storageClassDeviceSets: []
          useAllDevices: true
          useAllNodes: true
          deviceFilter: "<<value>>"
...
```

   b. To assign specific storage devices on all nodes to the Ceph cluster:

```
...
rook-ceph-cluster:
    enabled: true
    values: |
      cephClusterSpec:
        storage:
          storageClassDeviceSets: []
          useAllNodes: true
          useAllDevices: false
          deviceFilter: "^sdb."
...
```

797 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
798 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/

Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[799]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[800].

5. *If required:* Customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, etc.
6. *If required:* If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
...
apps:
...
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true
...
```

7. Expand **one** of the following sets of instructions, depending on your license and application environments:

Essential License: Install Kommander in a Pre-provisioned, Non-Air-Gapped Environment

6.11.4.5.1.2 Essential License: Install Kommander

Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Enterprise License: Install Kommander in a Pre-provisioned, Non-air-gapped Environment with DKP Catalog Applications

799 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
800 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

6.11.4.5.1.3 Enterprise License: Install Kommander with DKP Catalog Applications

If you want to enable DKP Catalog applications after installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.7.0
...
```

⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

6.11.4.5.2 Kommander Customizations

You can configure the Kommander component of DKP during the initial installation, and also post-installation using the DKP CLI. If you are not sure of what you want to customize during install, then proceed to the next step. To read about Kommander component customization options, refer to this section of the documetation: Kommander Customizations (see page 1558)

6.11.4.5.3 Next Step

Day 2 - Cluster Operations Management (see page 590)

6.11.5 Pre-provisioned Install in an Air-gapped Environment

In pre-provisioned environments, DKP handles your cluster's lifecycle including installation, upgrade, and node management. DKP installs Kubernetes, monitoring and logging apps, as well as its own user interface.

In an environment with access to the Internet, you retrieve artifacts from specialized repositories dedicated to them, such as Docker® images contained in DockerHub, and Helm™ Charts that come from a dedicated Helm Chart repository. However, in an air-gapped environment, you need local repositories to store Helm charts, Docker images, and other artifacts. Tools such as JFrog™, Harbor™, and Nexus™ handle multiple types of artifacts in one local repository.

6.11.5.1 Installation Process

6.11.5.2 Pre-provisioned Air-gapped Define Environment

6.11.5.2.1 Fulfill the prerequisites for using a pre-provisioned infrastructure when Air-Gapped

The instructions below outline how to fulfill the prerequisites for using pre-provisioned infrastructure when using an air-gapped environment.

6.11.5.2.2 Air-Gapped Registry Prerequisites

DKP in an air-gapped environment requires a local container registry (see page 1606) of trusted images to enable production level Kubernetes cluster management. In an environment with access to the internet, you retrieve artifacts from specialized repositories dedicated to them such as Docker images contained in DockerHub and Helm Charts that come from a dedicated Helm Chart repository. However, in an air-gapped environment, you need:

- Local repositories (see page 1606) to store Helm charts, Docker images and other artifacts. Tools such as ECR, jFrog, Harbor and Nexus handle multiple types of artifacts in one local repository.
- Bastion Host - If you have not set up a Bastion Host (see page 1068) yet, refer to that section of the Documentation.
- The complete DKP air-gapped bundle which contains all the DKP components needed for an air-gapped environment installation and also to use a local registry in a non-air-gapped environment: Pre-provisioned Loading the Registry (see page 1076)

6.11.5.2.2.1 Copy Air-gapped Artifacts onto Cluster Hosts

Using the Konvoy Image Builder (see page 1626), you can copy the required artifacts (such as charts, java or OS packages like RPM or Deb) onto your cluster hosts.

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` and extracted the tarball to a local directory above, the Kubernetes image bundle is located in `kib/artifacts/images`. To verify images and artifacts are have extracted there:
   a. Verify the image bundles exist in `kib/artifacts/images`:

```
$ ls kib/artifacts/images/
kubernetes-images-1.28.7-d2iq.1.tar kubernetes-images-1.28.7-d2iq.1-
fips.tar
```

   b. Verify the artifacts for your OS exist in the `artifacts/` directory and export the appropriate variables:

```
$ ls kib/artifacts/
1.28.7_centos_7_x86_64.tar.gz       1.28.7_redhat_8_x86_64_fips.tar.gz
containerd-1.6.28-d2iq.1-rhel-7.9-x86_64.tar.gz   containerd-1.6.28-
d2iq.1-rhel-8.6-x86_64_fips.tar.gz  pip-packages.tar.gz
1.28.7_centos_7_x86_64_fips.tar.gz  1.28.7_rocky_9_x86_64.tar.gz
containerd-1.6.28-d2iq.1-rhel-7.9-x86_64_fips.tar.gz  containerd-1.6.28-
d2iq.1-rocky-9.0-x86_64.tar.gz
1.28.7_redhat_7_x86_64.tar.gz       1.28.7_ubuntu_20_x86_64.tar.gz
containerd-1.6.28-d2iq.1-rhel-8.4-x86_64.tar.gz   containerd-1.6.28-
d2iq.1-rocky-9.1-x86_64.tar.gz
```

```
1.28.7_redhat_7_x86_64_fips.tar.gz  containerd-1.6.28-d2iq.1-centos-7.9-
x86_64.tar.gz       containerd-1.6.28-d2iq.1-rhel-8.4-x86_64_fips.tar.gz
containerd-1.6.28-d2iq.1-ubuntu-20.04-x86_64.tar.gz
1.28.7_redhat_8_x86_64.tar.gz       containerd-1.6.28-d2iq.1-centos-7.9-
x86_64_fips.tar.gz  containerd-1.6.28-d2iq.1-rhel-8.6-x86_64.tar.gz
images
```

c. For example, for RHEL 8.4 you would set:

```
export OS_PACKAGES_BUNDLE=1.28.7_redhat_8_x86_64.tar.gz
export CONTAINERD_BUNDLE=containerd-1.6.28-d2iq.1-rhel-8.4-x86_64.tar.gz
```

2. Export the following environment variables, ensuring that all control plane and worker nodes are included:

```
export CONTROL_PLANE_1_ADDRESS="<control-plane-address-1>"
export CONTROL_PLANE_2_ADDRESS="<control-plane-address-2>"
export CONTROL_PLANE_3_ADDRESS="<control-plane-address-3>"
export WORKER_1_ADDRESS="<worker-address-1>"
export WORKER_2_ADDRESS="<worker-address-2>"
export WORKER_3_ADDRESS="<worker-address-3>"
export WORKER_4_ADDRESS="<worker-address-4>"
export SSH_USER="<ssh-user>"
export SSH_PRIVATE_KEY_FILE="<private key file>"
```

`SSH_PRIVATE_KEY_FILE` must be either the name of the SSH private key file in your working directory or an absolute path to the file in your user's home directory.

3. Generate an `inventory.yaml` to tell `konvoy-image` what the IP addresses are for the nodes in your cluster, so it knows where to upload the artifacts. This YAML is automatically picked up by the `konvoy-image upload` command in the next step. This `inventory.yaml` **should exclude any GPU workers**, which will be handled in final additional steps.

```
cat <<EOF > inventory.yaml
all:
  vars:
    ansible_user: $SSH_USER
    ansible_port: 22
    ansible_ssh_private_key_file: $SSH_PRIVATE_KEY_FILE
  hosts:
    $CONTROL_PLANE_1_ADDRESS:
      ansible_host: $CONTROL_PLANE_1_ADDRESS
    $CONTROL_PLANE_2_ADDRESS:
      ansible_host: $CONTROL_PLANE_2_ADDRESS
    $CONTROL_PLANE_3_ADDRESS:
      ansible_host: $CONTROL_PLANE_3_ADDRESS
    $WORKER_1_ADDRESS:
```

```
      ansible_host: $WORKER_1_ADDRESS
    $WORKER_2_ADDRESS:
      ansible_host: $WORKER_2_ADDRESS
    $WORKER_3_ADDRESS:
      ansible_host: $WORKER_3_ADDRESS
    $WORKER_4_ADDRESS:
      ansible_host: $WORKER_4_ADDRESS
EOF
```

4. Upload the artifacts onto cluster hosts with the following command:

```
konvoy-image upload artifacts \
              --container-images-dir=./kib/artifacts/images/ \
              --os-packages-bundle=./kib/artifacts/$OS_PACKAGES_BUNDLE \
              --containerd-bundle=./kib/artifacts/$CONTAINERD_BUNDLE \
              --pip-packages-bundle=./kib/artifacts/pip-packages.tar.gz
```

The `konvoy-image upload artifacts` command copies all OS packages and other artifacts onto each of the machines in your inventory. When you create the cluster, the provisioning process connects to each node and runs commands to install those artifacts and consequently Kubernetes running. KIB uses variable overrides (see page 1670) to specify base image and container images to use in your new machine image. The variable overrides files for NVIDIA and FIPS can be ignored unless adding an overlay feature.

- Use the overrides flag (EX: `--overrides overrides/fips.yaml`) and reference either `fips.yaml` or `offline-fips.yaml` manifests located in the overrides directory[801] or see these pages in the documentation:
  - FIPS Overrides (see page 1671)
  - Create FIPS 140 Images (see page 1602)

GPU Only Steps

Additional steps are required if using GPU:

If the NVIDIA runfile[802] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory.

801 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides
802 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile

- ```
  curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/NVIDIA-Linux-x86_64-470.82.01.run
  mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
  ```

1. Create an inventory for GPU Nodes.

```
cat <<EOF >  gpu_inventory.yaml
all:
  vars:
    ansible_port: 22
    ansible_ssh_private_key_file: $SSH_PRIVATE_KEY_FILE
    ansible_user: $SSH_USER

  hosts:
    $GPU_WORKER_1_ADDRESS:
      ansible_host: $GPU_WORKER_1_ADDRESS
EOF
```

2. Upload the artifacts to the gpu nodepool with the `nvidia-runfile` flag

```
konvoy-image upload artifacts \
    --inventory-file=gpu_inventory.yaml \
    --container-images-dir=./kib/artifacts/images/ \
    --os-packages-bundle=./kib/artifacts/$OS_PACKAGES_BUNDLE \
    --containerd-bundle=./kib/artifacts/$CONTAINERD_BUNDLE \
    --pip-packages-bundle=./kib/artifacts/pip-packages.tar.gz \
    --nvidia-runfile=./kib/artifacts/NVIDIA-Linux-x86_64-470.82.01.run
```

6.11.5.2.2.2 Next Step

Pre-provisioned Air-gapped Bootstrap Cluster (see page 1124)

6.11.5.3 Pre-provisioned Air-gapped Bootstrap Cluster

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster. The workload cluster then manages its own lifecycle.
```

To create Kubernetes clusters, Konvoy uses Cluster API[803] (CAPI) controllers. These controllers run on a Kubernetes cluster. To get started, you need a *bootstrap* cluster. By default, Konvoy creates a bootstrap cluster for you in a Docker container using the Kubernetes-in-Docker (KIND[804]) tool.

6.11.5.3.1 Prerequisites

Before you begin, you must:

- Complete the steps in Prerequisites (see page 141).
- Ensure the `dkp` binary can be found in your $PATH.
- IF using a Registry Mirror even though you are not in an air-gapped environment, refer to the air-gapped section for loading images: Pre-provisioned Air-gapped Define Environment (see page 1120)

6.11.5.3.1.1 Bootstrap Cluster Lifecycle Services

1. Review Universal Configurations for all Infrastructure Providers (see page 1052) regarding settings, flags and other choices and then begin bootstrapping.
2. Create a bootstrap cluster:

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config
   ```

   - Configuring an HTTP/HTTPS Proxy (see page 1053) use `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful.

HTTP Flags if needed:

To create a bootstrap cluster in a proxied environment (see page 1054) use this command syntax, in addition to any other flags you may need:

```
--http-proxy <string> \
--https-proxy <string> \
--no-proxy <string>
```

Output:

```
✓ Creating a bootstrap cluster
✓ Initializing new CAPI components
```

Konvoy creates a bootstrap cluster using KIND[805] as a library and deploys Cluster API[806] providers on the cluster. Refer to Customizing CAPI Components for a Cluster (see page 1061)for more details.

803 https://cluster-api.sigs.k8s.io/
804 https://github.com/kubernetes-sigs/kind
805 https://github.com/kubernetes-sigs/kind
806 https://cluster-api.sigs.k8s.io/

Konvoy waits until the controller-manager and webhook deployments of these providers are ready. List these deployments using this command:

```
kubectl get --all-namespaces deployments -l=clusterctl.cluster.x-k8s.io
```

Output:

```
NAMESPACE                           NAME
READY   UP-TO-DATE   AVAILABLE   AGE
capa-system                         capa-controller-manager                         1
/1      1            1           2m8s
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-manager       1
/1      1            1           2m10s
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-manager   1
/1      1            1           2m10s
capi-system                         capi-controller-manager                         1
/1      1            1           2m11s
cappp-system                        cappp-controller-manager                        1
/1      1            1           2m6s
capv-system                         capv-controller-manager                         1
/1      1            1           2m5s
capz-system                         capz-controller-manager                         1
/1      1            1           2m7s
cert-manager                        cert-manager                                    1
/1      1            1           2m21s
cert-manager                        cert-manager-cainjector                         1
/1      1            1           2m21s
cert-manager                        cert-manager-webhook                            1
/1      1
```

4. Konvoy creates a bootstrap cluster using KIND[807] as a library. Konvoy then deploys the following Cluster API[808] providers on the cluster:

- Core Provider[809]
- vSphere Infrastructure Provider[810]
- Kubeadm Bootstrap Provider[811]
- Kubeadm ControlPlane Provider[812]

6.11.5.3.1.2 Next Step

Pre-provisioned Air-gapped New Cluster (see page 1127)

807 https://github.com/kubernetes-sigs/kind
808 https://cluster-api.sigs.k8s.io/
809 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/
810 https://github.com/kubernetes-sigs/cluster-api-provider-vsphere
811 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/bootstrap/kubeadm
812 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/controlplane/kubeadm

6.11.5.4 Pre-provisioned Air-gapped New Cluster

If your cluster is in an air-gapped environment, you must provide additional arguments when creating the cluster. Before you create a new DKP cluster below, other customizations are available that require different flags during `dkp create cluster` command. Refer to Pre-provisioned Cluster Creation Customization Choices (see page 1083) for more cluster customization options.

6.11.5.4.1 Name your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[813] for more naming information.

When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects (see page 1074).

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<preprovisioned-example>
```

Before you create a new DKP cluster below, choose an external load balancer or virtual IP (see page 1100) and use the corresponding `dkp create cluster` command flags.

6.11.5.4.2 Create an Air-gapped Kubernetes Cluster

Once you've defined the infrastructure and control plane endpoints, you can proceed to creating the cluster by following these steps to create a new pre-provisioned cluster.

813 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

> ⚠ DKP uses `localvolumeprovisioner` as the default storage provider (see page 110) for a pre-provisioned environment. However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[814] compatible storage that is suitable for production.

After disabling `localvolumeprovisioner`, you can choose from any of the storage options available for Kubernetes. To make that storage the default storage, use the commands shown in this section of the Kubernetes documentation: https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

The following command relies on the pre-provisioned cluster API infrastructure provider to initialize the Kubernetes control plane and worker nodes on the hosts defined in the inventory.

- **NOTE:** To increase Docker Hub's rate limit[815] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.
- **NOTE:** Ensure your subnets (see page 1102) do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

Create cluster command - Depending on the cluster size, it will take a few minutes to create:

1. The command below uses the **default external load balancer** to create Kubernetes cluster objects: **(Optional)** If you have overrides for your clusters[816], you must specify the secret as part of the create cluster command. If these are not specified, the overrides for your nodes will not be applied. `--override-secret-name=$CLUSTER_NAME-user-overrides`

```
dkp create cluster preprovisioned --cluster-name ${CLUSTER_NAME}
  --control-plane-endpoint-host <control plane endpoint host>
  --control-plane-endpoint-port <control plane endpoint port, if different than
6443>
```

814 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
815 https://docs.docker.com/docker-hub/download-rate-limit/
816 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29918736

```
  --pre-provisioned-inventory-file preprovisioned_inventory.yaml
  --ssh-private-key-file <path-to-ssh-private-key>
  --registry-mirror-url=${REGISTRY_URL} \
  --dry-run \
  --output=yaml \
  > ${CLUSTER_NAME}.yaml
```

a. **Virtual IP ALTERNATIVE** - if you don't have an external LB, and wish to use a **VIRTUAL IP** provided by kube-vip, specify these flags example below:

```
dkp create cluster preprovisioned \
    --cluster-name ${CLUSTER_NAME} \
    --control-plane-endpoint-host 196.168.1.10 \
    --virtual-ip-interface eth1 \
    --dry-run \
    --output=yaml \
    > ${CLUSTER_NAME}.yaml
```

HTTP only:

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

```
--http-proxy <<http proxy list>>
--https-proxy <<https proxy list>>
--no-proxy <<no proxy list>>
```

- To configure the Control Plane and Worker nodes to use an HTTP proxy:

```
export CONTROL_PLANE_HTTP_PROXY=http://example.org:8080
export CONTROL_PLANE_HTTPS_PROXY=http://example.org:8080
export
CONTROL_PLANE_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1
,10.96.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.s
vc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.sv
c.cluster,.svc.cluster.local,169.254.169.254"

export WORKER_HTTP_PROXY=http://example.org:8080
export WORKER_HTTPS_PROXY=http://example.org:8080
export
WORKER_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96.
0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kube
rnetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.clust
er,.svc.cluster.local,169.254.169.254"
```

- Replace:

- `example.org,example.com,example.net` with you internal addresses
- `localhost` and `127.0.0.1` addresses should not use the proxy
- `10.96.0.0/12` is the default Kubernetes service subnet
- `192.168.0.0/16` is the default Kubernetes pod subnet
- `kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local` is the internal Kubernetes kube-apiserver service
- `.svc,.svc.cluster,.svc.cluster.local` is the internal Kubernetes services

- Use the flags in the `dkp create cluster` command:

```
--control-plane-http-proxy=${CONTROL_PLANE_HTTP_PROXY} \
--control-plane-https-proxy=${CONTROL_PLANE_HTTPS_PROXY} \
--control-plane-no-proxy=${CONTROL_PLANE_NO_PROXY} \
--worker-http-proxy=${WORKER_HTTP_PROXY} \
--worker-https-proxy=${WORKER_HTTPS_PROXY} \
--worker-no-proxy=${WORKER_NO_PROXY} \
```

Additional Registry flags if needed:

Set the flag during cluster creation from exported variables on the bootstrap (see page 1216) page:

```
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD}
```

See also: Use a Registry Mirror (see page 1609)

FIPS Requirements

To create a cluster in FIPS mode (see page 1598), inform the controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes by adding those flags to `dkp create cluster` command:

```
--kubernetes-version=v1.28.7+fips.0 \
--etcd-version=3.5.10+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
```

Individual manifests using the Output Directory flag:

You can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag. This will create multiple files in the specified directory which must already exist:

```
--output-directory=<existing-directory>
```

Refer to the Cluster Creation Customization Choices (see page 1083) section for more information on how to use optional flags such as the `--output-directory` flag.

The output from this command is shortened here for reading clarity, but should start like this:

```
Generating cluster resources
cluster.cluster.x-k8s.io/preprovisioned-example created
cont.........
```

2. Inspect or edit the cluster objects and familiarize yourself with Cluster API before editing the cluster objects as edits can prevent the cluster from deploying successfully. See Pre-provisioned Customizing CAPI Clusters (see page 1085) .

Familiarize yourself with Cluster API before editing the cluster objects as edits can prevent the cluster from deploying successfully.

3. Create the cluster from the objects generated in the `dry run`. A warning will appear in the console if the resource already exists and will require you to remove the resource or update your YAML.

   ```
   kubectl create -f ${CLUSTER_NAME}.yaml
   ```

   NOTE: If you used the `--output-directory` flag in your `dkp create .. --dry-run` step above, create the cluster from the objects you created by specifying the directory:

   ```
   kubectl create -f <existing-directory>/
   ```

4. Use the wait command to monitor the cluster control-plane readiness:

   ```
   kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
   timeout=30m
   ```

   Output:

   ```
   cluster.cluster.x-k8s.io/preprovisioned-example condition met
   ```

NOTE: Depending on the cluster size, it will take a few minutes to create.

When the command completes, you will have a running Kubernetes cluster! Use this command to get the Kubernetes `kubeconfig` for the new cluster and proceed to installing the DKP Kommander UI:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

Azure requires changing the CNI encapsulation type of Calico from the default of IPtoIP to VXlan. If changing the Calico encapsulation (see page 1095), D2iQ recommends changing it after cluster creation, but before production.

6.11.5.4.3 Audit logs

To modify Control Plane Audit logs settings using the information contained in the page Configure the Control Plane (see page 1613).

6.11.5.4.3.1 Further Optional Steps:

- Azure Only Configurations (see page 1079)
- Modify the Calico installation (see page 1095)

6.11.5.4.3.2 Cluster Verification

If you want to **monitor** or verify the installation of your clusters, refer to Verify your Cluster and DKP Installation (see page 1622).

6.11.5.4.3.3 Next Step:

Pre-provisioned Make your Air-gapped Cluster Self-managed (see page 1132)

6.11.5.5 Pre-provisioned Make your Air-gapped Cluster Self-managed

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which then deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, which is now self-managed. This guide describes how to make a workload cluster self-managed.

This page contains instructions on how to make your cluster self-managed. This is necessary if there is only one cluster in your environment, or if this cluster should become the Management cluster in a multi-cluster environment.

If you already have a self-managed or Management cluster in your environment, skip this page.

6.11.5.5.1 Make the New Kubernetes Cluster Manage Itself

Your new cluster is turned into a **Management Cluster** (or free standing Essential Cluster) using this procedure:

If you have not already retrieved the kubeconfig after creating the cluster, use this command before proceeding: `dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf`

1. Deploy cluster lifecycle services on the workload cluster:

   ```
   dkp create capi-components --kubeconfig ${CLUSTER_NAME}.conf
   ```

   Output:

   ```
   ✓ Initializing new CAPI components
   ```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

2. Move the Cluster API objects from the bootstrap to the workload cluster:

   The cluster lifecycle services on the workload cluster are ready, but the workload cluster configuration is on the bootstrap cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the bootstrap to the workload cluster. This process is also called a Pivot[817]. First unset the kubeconfig and then move the CAPI:

   ```
   unset KUBECONFIG
   ```

   Next:

   ```
   dkp move capi-resources --to-kubeconfig ${CLUSTER_NAME}.conf
   ```

817 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

Output:

```
✓ Moving cluster resources
You can now view resources in the moved cluster by using the --kubeconfig flag
with kubectl. For example: kubectl --kubeconfig=preprovisioned-example.conf get
nodes
```

NOTE: To ensure only one set of cluster lifecycle services manages the workload cluster, Konvoy first pauses reconciliation of the objects on the bootstrap cluster, then creates the objects on the workload cluster. As Konvoy copies the objects, the cluster lifecycle services on the workload cluster reconcile the objects. The workload cluster becomes self-managed after Konvoy creates all the objects. If it fails, the `move` command can be safely retried.

3. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf wait --for=condition=ControlPlaneRead
y "clusters/${CLUSTER_NAME}" --timeout=20m
```

Output:

```
cluster.cluster.x-k8s.io preprovisioned-example condition met
```

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use Konvoy with the workload cluster kubeconfig.

```
dkp describe cluster --kubeconfig ${CLUSTER_NAME}.conf -c ${CLUSTER_NAME}
```

Output:

```
NAME
READY  SEVERITY  REASON  SINCE  MESSAGE
Cluster/preprovisioned-example                                              True
2m31s
├─ClusterInfrastructure - PreprovisionedCluster/preprovisioned-example
├─ControlPlane - KubeadmControlPlane/preprovisioned-example-control-plane  True
2m31s
│ ├─Machine/preprovisioned-example-control-plane-6g6nr                      True
2m33s
│ ├─Machine/preprovisioned-example-control-plane-8lhcv                      True
2m33s
│ └─Machine/preprovisioned-example-control-plane-kk2kg                      True
2m33s
└─Workers
```

```
  └─MachineDeployment/preprovisioned-example-md-0                    True
2m34s
    └─Machine/preprovisioned-example-md-0-77f667cd9-tnctd            True
2m33s
```

5. Remove the bootstrap cluster, as the workload cluster is now self-managed:

```
dkp delete bootstrap
```

Output:

```
✓ Deleting bootstrap cluster
```

6.11.5.5.1.1 Known limitations

Be aware of these limitations in the current release of Konvoy.

- DKP supports moving only one set of cluster objects from the bootstrap cluster to the workload cluster, or vice-versa.
- DKP only supports moving all namespaces in the cluster; DKP does not support migration of individual namespaces.

6.11.5.5.1.2 Next Step

Pre-provisioned Install Kommander in an Air-gapped Environment (see page 1135)

6.11.5.6 Pre-provisioned Install Kommander in an Air-gapped Environment

This section provides installation instructions for the Kommander component of DKP in an air-gapped pre-provisioned environment.

6.11.5.6.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry: Air-gapped Environments (see page 1536).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

6.11.5.6.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

```
export CLUSTER_NAME=<your-management-cluster-name>
```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1558) for the deployment:

```
dkp install kommander --init --airgapped > kommander.yaml
```

4. *If required*, **customize** your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include:
   - Custom Domains and Certificates
   - HTTP proxy
   - External Load Balancer
   - GPU utilization, etc.
   - Rook Ceph customization for Pre-provisioned environments (see page 1541)
5. *If required*: If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
...
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true
...
```

6. Expand **one** of the following sets of instructions, depending on your license and application environments:

Essential License: Install Kommander in a Pre-provisioned, Air-Gapped Environment

6.11.5.6.1.2 Essential License: Install Kommander in an Air-gapped Environment

Use the customized `kommander.yaml` to install DKP in a pre-provisioned, air-gapped environment:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.0.tar.gz
```

Enterprise License: Install Kommander in a Pre-provisioned, Air-gapped Environment with DKP Catalog Applications

6.11.5.6.1.3 Enterprise License: Install Kommander in an Air-gapped Environment with DKP Catalog Applications

If you want to enable DKP Catalog applications **AFTER** installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add the following values to enable DKP Catalog Applications:

   ```
   ...
   catalog:
     repositories:
       - name: dkp-catalog-applications
         labels:
           kommander.d2iq.io/project-default-catalog-repository: "true"
           kommander.d2iq.io/workspace-default-catalog-repository: "true"
           kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
         path: ./dkp-catalog-applications-v2.8.0.tar.gz
   ...
   ```

   ⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

   ```
   dkp install kommander --installer-config kommander.yaml --kubeconfig=$
   {CLUSTER_NAME}.conf \
   ```

```
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.0.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.8.0.tar.gz
```

6.11.5.6.2 Kommander Customizations

You can configure the Kommander component of DKP during the initial installation, and also post-installation using the DKP CLI. If you are not sure of what you want to customize during install, then proceed to the next step. To read about Kommander component customization options, refer to this section of the documetation: Kommander Customizations (see page 1558)

6.11.5.6.3 Next Step

Day 2 - Cluster Operations Management (see page 590)

6.11.6 Pre-provisioned Management Tools

After cluster creation and configuration, you can revisit clusters to update and change variables.

- Pre-provisioned Delete Cluster (see page 1138)
- Pre-provisioned Manage Node Pools (see page 1141)

6.11.6.1 Pre-provisioned Delete Cluster

NOTE: A self-managed workload cluster cannot delete itself. If your workload cluster is self-managed, you must create a bootstrap cluster and move the cluster lifecycle services to the bootstrap cluster before deleting the workload cluster.

If you did not make your workload cluster self-managed, as described in Make New Cluster Self-Managed (see page 1113), see Delete the workload cluster (see page 1140).

6.11.6.1.1 Prepare to Delete the Pre-provisioned Cluster

Follow these steps:

1. Create a bootstrap cluster:

   The bootstrap cluster will host the Cluster API controllers that reconcile the cluster objects marked for deletion:

NOTE: To avoid using the wrong kubeconfig, the following steps use explicit kubeconfig paths and contexts.

```
dkp create bootstrap --kubeconfig $HOME/.kube/config
```

```
 ✓ Creating a bootstrap cluster
 ✓ Initializing new CAPI components
```

2. Move the Cluster API objects from the workload to the bootstrap cluster: The cluster lifecycle services on the bootstrap cluster are ready, but the workload cluster configuration is on the workload cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the workload to the bootstrap cluster. This process is also called a Pivot[818].

```
dkp move capi-resources \
    --from-kubeconfig ${CLUSTER_NAME}.conf \
    --from-context ${CLUSTER_NAME}-admin@${CLUSTER_NAME} \
    --to-kubeconfig $HOME/.kube/config \
    --to-context kind-konvoy-capi-bootstrapper
```

```
✓ Moving cluster resources
You can now view resources in the moved cluster by using the --kubeconfig flag
with kubectl. For example: kubectl --kubeconfig $HOME/.kube/config get nodes
```

3. Use the cluster lifecycle services on the workload cluster to check the workload cluster status by running the following command:

```
dkp describe cluster --kubeconfig $HOME/.kube/config -c ${CLUSTER_NAME}
```

```
NAME
READY  SEVERITY  REASON  SINCE  MESSAGE
Cluster/preprovisioned-example                                          True
2m31s
├─ClusterInfrastructure - PreprovisionedCluster/preprovisioned-example
```

818 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
├─ControlPlane - KubeadmControlPlane/preprovisioned-example-control-plane  True
2m31s
│ ├─Machine/preprovisioned-example-control-plane-6g6nr                     True
2m33s
│ ├─Machine/preprovisioned-example-control-plane-8lhcv                     True
2m33s
│ └─Machine/preprovisioned-example-control-plane-kk2kg                     True
2m33s
└─Workers
  └─MachineDeployment/preprovisioned-example-md-0                          True
2m34s
    └─Machine/preprovisioned-example-md-0-77f667cd9-tnctd                  True
2m33s
```

NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use dkp with the workload cluster kubeconfig.

4. Wait for the cluster control-plane to be ready. Run the command below and wait for the condition to be met:

```
kubectl --kubeconfig $HOME/.kube/config wait --for=condition=controlplaneready
"clusters/${CLUSTER_NAME}" --timeout=20m
```

```
cluster.cluster.x-k8s.io/preprovisioned-example condition met
```

Persistent Volumes (PVs) are not deleted automatically by design in order to preserve your data. However, they take up storage space if not deleted. You must delete PVs manually. Information for backup of a cluster and PVs is on the page in documentation called Back up your Cluster's Applications and Persistent Volumes (see page 863) .

6.11.6.1.2 Delete the Workload Cluster

If you have a need to remove the Kubernetes cluster, such as for environment cleanup, use this command to delete the provisioned Kubernetes cluster.

1. To delete a cluster, you would use `dkp delete cluster` and pass in the name of the cluster you are trying to delete with `--cluster-name` flag. You would use `kubectl get clusters` to get those details (`--cluster-name` and `--namespace`) of the Kubernetes cluster to delete it.

NOTE: Do not use `dkp get clusters` since that gets you Kommander cluster details rather than Konvoy kubernetes cluster details.

```
kubectl get clusters
```

2. Delete the Kubernetes cluster and wait a few minutes:

   NOTE: Before deleting the cluster, DKP deletes all Services of type LoadBalancer on the cluster. Each Service is backed by an AWS Classic ELB. Deleting the Service deletes the ELB that backs it. To skip this step, use the flag `--delete-kubernetes-resources=false`. Do not skip this step if the VPC is managed by DKP. When DKP deletes the cluster, it deletes the VPC. If the VPC has any AWS Classic ELBs, AWS does not allow the VPC to be deleted, and DKP cannot delete the cluster.

   ```
   dkp delete cluster --cluster-name=${CLUSTER_NAME} --kubeconfig $HOME/.kube/
   config
   ```

   ```
   ✓ Deleting Services with type LoadBalancer for Cluster default/preprovisioned-
   example
   ✓ Deleting ClusterResourceSets for Cluster default/preprovisioned-example
   ✓ Deleting cluster resources
   ✓ Waiting for cluster to be fully deleted
   Deleted default/preprovisioned-example cluster
   ```

6.11.6.1.3 Delete the Bootstrap Cluster

After you have moved the workload resources back to a bootstrap cluster and deleted the workload cluster, you no longer need the bootstrap cluster. You can safely delete the bootstrap cluster with this command:

Use `dkp` with the bootstrap cluster to delete the workload cluster. Delete the `kind` Kubernetes cluster:

```
dkp delete bootstrap --kubeconfig $HOME/.kube/config
```

```
✓ Deleting bootstrap cluster
```

6.11.6.2 Pre-provisioned Manage Node Pools

Node pools are part of a cluster and managed as a group, and you can use a node pool to manage a group of machines using the same common properties. When Konvoy creates a new default cluster, there is one node pool for the worker nodes and all nodes in that new node pool have the same configuration. You can create additional node pools for more specialized hardware or configuration. For example, if you want to tune your memory usage on a cluster where you need maximum memory for some machines and minimal memory on other machines, you would create a new node pool with those specific resource needs.

DKP implements node pools using Cluster API MachineDeployments[819].

- Pre-provisioned Create and Delete Node Pools (see page 1142)
- Pre-provisioned Add Nodes to Existing Node Pool (see page 1144)
- Pre-provisioned GPU Nodepools (see page 1146)

6.11.6.2.1 Pre-provisioned Create and Delete Node Pools

Node pools are part of a cluster and managed as a group, and can be used to manage a group of machines using common properties. New default clusters created by Konvoy contain one node pool of worker nodes that have the same configuration.

You can create additional node pools for specialized hardware or other configurations. For example, if you want to tune your memory usage on a cluster where you need maximum memory for some machines and minimal memory on others, you could create a new node pool with those specific resource needs.

NOTE: Konvoy implements node pools using Cluster API MachineDeployments[820].

6.11.6.2.1.1 Create a Pre-provisioned Node Pool

Follow these steps:

1. Create an inventory object that has the same name as the node pool you're creating, and the details of the pre-provisioned machines that you want to add to it. For example, to create a node pool named `gpu-nodepool` an inventory named `gpu-nodepool` must be present in the same namespace:

```
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: ${MY_NODEPOOL_NAME}
spec:
  hosts:
    - address: ${IP_OF_NODE}
  sshConfig:
    port: 22
    user: ${SSH_USERNAME}
    privateKeyRef:
      name: ${NAME_OF_SSH_SECRET}
      namespace: ${NAMESPACE_OF_SSH_SECRET}
```

819 https://cluster-api.sigs.k8s.io/developer/architecture/controllers/machine-deployment.html
820 https://cluster-api.sigs.k8s.io/developer/architecture/controllers/machine-deployment.html

2. (Optional) If your pre-provisioned machines have overrides (see page 1670), you must create a secret that includes all of the overrides you want to provide in one file. Create an override secret (see page 1086) using the instructions detailed on this page.
3. Once the `PreprovisionedInventory` object and overrides are created, create a node pool:

```
dkp create nodepool preprovisioned -c ${MY_CLUSTER_NAME} ${MY_NODEPOOL_NAME} --
override-secret-name ${MY_OVERRIDE_SECRET}
```

- Advanced users can use a combination of the `--dry-run` and `--output=yaml` or `--output-directory=<existing-directory>` flags to get a complete set of node pool objects to modify locally or store in version control.

> For more information regarding this flag or others, please refer to the dkp create nodepool (see page 1875) section of the documentation for either cluster or nodepool and select your provider.

6.11.6.2.1.2 Delete a Node Pool

Deleting a node pool deletes both the Kubernetes nodes and their underlying infrastructure. DKP drains all nodes prior to deletion and reschedules the pods running on those nodes.

To delete a node pool from a managed cluster, run the following command:

```
dkp delete nodepool ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME}
```

The expected output is similar to the following example, indicating the `example` node pool is being deleted:

```
INFO[2021-07-28T17:14:26-07:00] Running nodepool delete command
Nodepool=example clusterName=d2iq-e2e-cluster-1 managementClusterKubeconfig=
namespace=default src="nodepool/delete.go:80"
```

Deleting an invalid node pool results in an output similar to this example command output:

```
dkp delete nodepool ${CLUSTER_NAME}-md-invalid --cluster-name=${CLUSTER_NAME}

INFO[2021-07-28T17:11:44-07:00] Running nodepool delete command
Nodepool=demo-cluster-md-invalid clusterName=d2iq-e2e-cluster-1
 managementClusterKubeconfig= namespace=default src="nodepool/delete.go:80"
Error: failed to get nodepool with name demo-cluster-md-invalid in namespace default
: failed to get nodepool with name demo-cluster-md-invalid in namespace default :
machinedeployments.cluster.x-k8s.io "demo-cluster-md-invalid" not found
```

Next Topic

Pre-provisioned Add Nodes to Existing Node Pool (see page 1144)

6.11.6.2.2 Pre-provisioned Add Nodes to Existing Node Pool

This sections covers the prerequisites and procedure you need to scale-up or scale-down nodes in an existing DKP cluster.

6.11.6.2.2.1 Prerequisites

- You must have the bootstrap node running with the SSH key/secrets created.
- The export values in the environment variables section should contain the addresses of the nodes that you need to add Pre-provisioned: Define Infrastructure (see page 149).
- Update the `preprovisioned_inventory.yaml` with the new host addresses.
- Run the `kubectl apply` command.

Scale up a Cluster Node

Follow these steps:

1. Fetch the existing `preprovisioned_inventory`:

   ```
   $ kubectl get preprovisionedinventory
   ```

2. Edit the `preprovisioned_inventory` to add additional IPs needed for additional worker nodes in the `spec.hosts:` section:

   ```
   $ kubectl edit preprovisionedinventory <preprovisioned_inventory> -n default
   ```

3. Add any additional IPs that you require:

   ```
   spec:
       hosts:
       - address: <worker.ip.add.1>
       - address: <worker.ip.add.2>
   ```

   After you edit `preprovisioned_inventory`, fetch the machine deployment. The naming convention with `md` means that it is for worker machines.
   For example:

```
$ kubectl --kubeconfig ${CLUSTER_NAME}.conf get machinedeployment

NAME                     CLUSTER        AGE     PHASE     REPLICAS   READY
UPDATED   UNAVAILABLE
machinedeployment-md-0   cluster-name   9m10s   Running   4          4       4
```

4. Scale the worker node to the required number. In this example we are scaling from 4 to 6 worker nodes:

```
$ kubectl --kubeconfig ${CLUSTER_NAME}.conf scale --replicas=6 machinedeploymen
t machinedeployment-md-0

machinedeployment.cluster.x-k8s.io/machinedeployment-md-0 scaled
```

5. Monitor the scaling with this command, by adding `-w` option to watch:

```
$ kubectl --kubeconfig ${CLUSTER_NAME}.conf get machinedeployment -w

NAME                     CLUSTER        AGE   PHASE       REPLICAS   READY
UPDATED   UNAVAILABLE
machinedeployment-md-0   cluster-name   20m   ScalingUp   6          4       6
       2
```

6. Also you can check the machine deployment if it is already scaled. The output should resemble this example:

```
$ kubectl --kubeconfig ${CLUSTER_NAME}.conf get machinedeployment

NAME                     CLUSTER        AGE     PHASE     REPLICAS   READY
UPDATED   UNAVAILABLE
machinedeployment-md-0   cluster-name   3h33m   Running   6          6       6
```

7. Alternately, you can use this command and verify the `NODENAME` column and you should see the additional worker nodes added and in `Running` state:

```
$ kubectl --kubeconfig ${CLUSTER_NAME}.conf get machines -o wide

NAME     CLUSTER     AGE     PROVIDERID     PHASE     VERSION     NODENAME
```

Scale Down a Cluster Node

Follow these steps

1. Run this command on your worker nodes:

```
kubectl scale machinedeployment <machinedeployment-name> --replicas <new
 number>
```

2. For control plane nodes, execute the following command:

```
kubectl scale kubeadmcontrolplane ${CLUSTER_NAME}-control-plane --replicas <new
 number>
```

Additional Notes for Scaling Down

It is possible for machines to get stuck in the provisioning stage when you scaling down. You can utilize a delete operation to clear the stale machine deployment:

```
kubectl delete machine ${CLUSTER_NAME}-control-plane-<hash>
```

```
kubectl delete machine <machinedeployment-name>-<hash>
```

Next Topic

GPU Nodepools in a Pre-provisioned Environment (see page 1146)

6.11.6.2.3 Pre-provisioned GPU Nodepools

For pre-provisioned environments, DKP has introduced the `nvidia-runfile` flag for Air-gapped Pre-provisioned environments. If the NVIDIA runfile[821] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory (you must create an `artifacts` directory if it doesn't already exist).

```
curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/NVIDIA-Linux-
x86_64-470.82.01.run
mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
```

821 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile

DKP supported NVIDIA driver[822] version is 470.x.

1. Create the secret that GPU nodepool would use, this secret is populated from the KIB overrides. In this example we have a file called, `overrides/nvidia.yaml`. It should resemble this:

```
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
```

2. Create a secret on the bootstrap cluster that is populated from the above file. We will name it `${CLUSTER_NAME}-user-overrides`

```
kubectl create secret generic ${CLUSTER_NAME}-user-overrides --from-
file=overrides.yaml=overrides/nvidia.yaml
```

3. Create an inventory and nodepool with the instructions below and use the `$CLUSTER_NAME-user-overrides` secret.

Follow these steps:

1. Create an inventory object that has the same name as the node pool you're creating, and the details of the pre-provisioned machines that you want to add to it. For example, to create a node pool named `gpu-nodepool` an inventory named `gpu-nodepool` must be present in the same namespace:

```
apiVersion: infrastructure.cluster.konvoy.d2iq.io/v1alpha1
kind: PreprovisionedInventory
metadata:
  name: ${MY_NODEPOOL_NAME}
spec:
  hosts:
    - address: ${IP_OF_NODE}
  sshConfig:
    port: 22
    user: ${SSH_USERNAME}
    privateKeyRef:
      name: ${NAME_OF_SSH_SECRET}
      namespace: ${NAMESPACE_OF_SSH_SECRET}
```

822 https://www.nvidia.com/Download/Find.aspx

2. (Optional) If your pre-provisioned machines have overrides (see page 1670), you must create a secret that includes all of the overrides you want to provide in one file. Create an override secret (see page 1086) using the instructions detailed on this page.
3. Once the `PreprovisionedInventory` object and overrides are created, create a node pool:

```
dkp create nodepool preprovisioned -c ${MY_CLUSTER_NAME} ${MY_NODEPOOL_NAME} --
override-secret-name ${MY_OVERRIDE_SECRET}
```

- Advanced users can use a combination of the `--dry-run` and `--output=yaml` or `--output-directory=<existing-directory>` flags to get a complete set of node pool objects to modify locally or store in version control.

For more information regarding this flag or others, please refer to the dkp create nodepool (see page 1875) section of the documentation for either cluster or nodepool and select your provider.

For more information, see:

- Pre-provisioned Create and Delete Node Pools (see page 1142)
- Pre-provisioned Air-gapped Define Environment (see page 1120)

6.12 AWS Infrastructure

6.12.1 Configuration Types

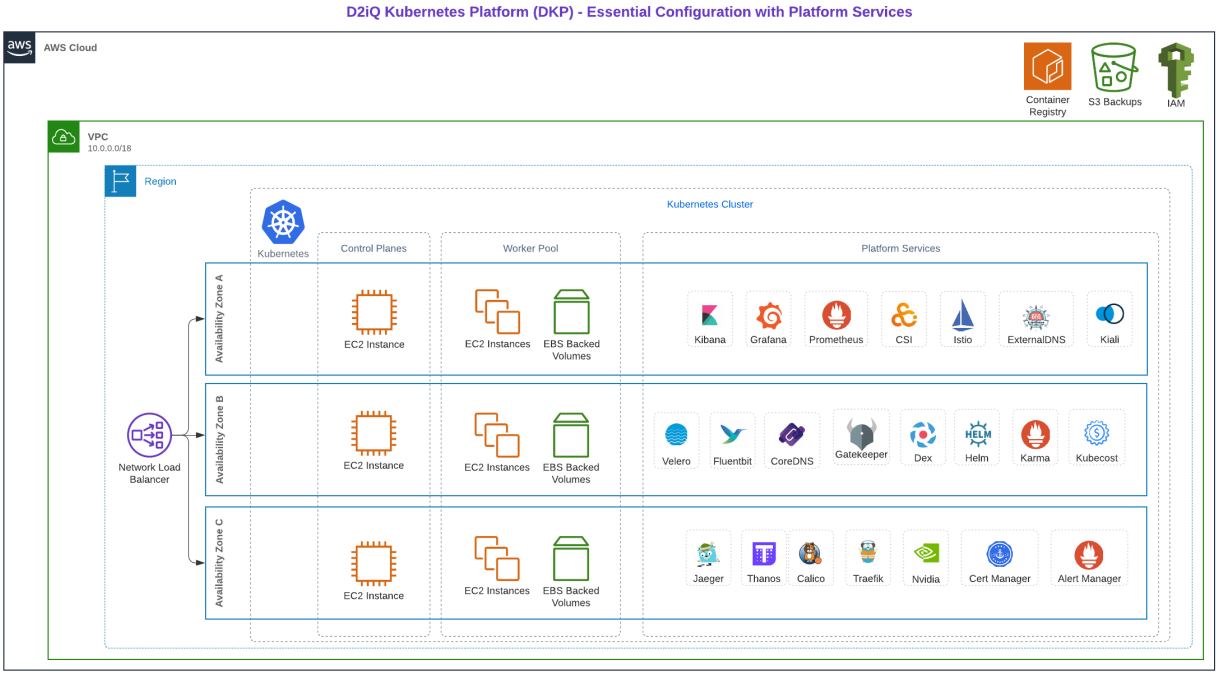
When installing DKP on AWS infrastructure, you can choose from multiple configuration types. The different types of AWS configuration types supported in DKP are listed below.

- AWS Prerequisites and Permissions (see page 1151)
- AWS Cluster Creation Customization Choices (see page 1170)
- AWS Install in a Non-air-gapped Environment (see page 1184)
- AWS Install in an Air-gapped Environment (see page 1207)
- AWS Management Tools (see page 1233)
- Configure Infrastructure in UI (see page 1258)

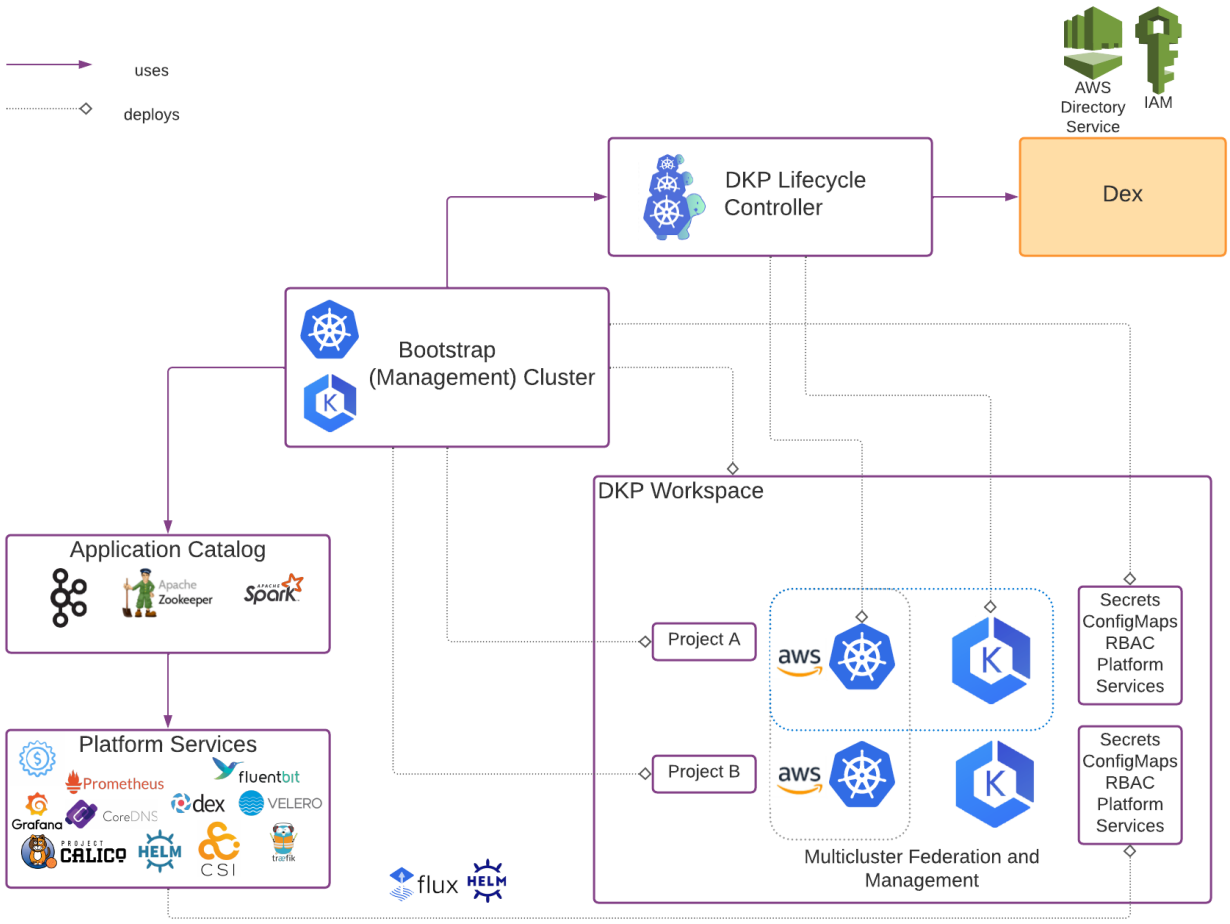
6.12.2 AWS Diagrams

The following diagrams show the two different ways you can implement DKP Essential and DKP Enterprise on AWS.

This diagram shows the granular detail of a single Kubernetes cluster running in AWS Cloud:



This diagram shows a higher-level view of DKP Enterprise, and assumes a multi-cluster environment, where each cluster might look like the single cluster example above:



6.12.3 AWS Pricing Considerations

Deploying AWS services can incur costs to your organization, depending on how and what you deploy. For more information, see the AWS Pricing Calculator[823].

6.12.4 AWS Service Limits

When using DKP on AWS, you need to be aware of the possibility of errors due to AWS service limits. For more information, see the AWS Service Limits[824].

6.12.5 Next Step:

AWS Prerequisites and Permissions (see page 1151)

823 https://calculator.aws/#/
824 https://aws.amazon.com/premiumsupport/knowledge-center/manage-service-limits/

6.12.6 AWS Prerequisites and Permissions

6.12.6.1 Prepare your environment to run DKP with AWS

Fulfilling the prerequisites involves completing these two areas:

1. DKP prerequisites
2. AWS prerequisites

6.12.6.2 1. DKP Prerequisites

Before you begin using Konvoy, you must have:

- An x86_64-based Linux or macOS machine.
- The `dkp` binary for Linux, or macOS.
- A Container engine/runtime installed is required to install DKP:
    - Version Docker®[825] container engine version 18.09.2 or higher installed for Linux or MacOS - On macOS, Docker runs in a virtual machine which needs configured with at least 8 GB of memory.
    - Version 4.0 of Podman[826] or higher for Linux. Host requirements found here: Host Requirements[827]
- kubectl[828] for interacting with the running cluster.
- A valid AWS account with credentials configured[829].
- For a local registry whether air-gapped or non-air-gapped environment, download and extract the bundle. Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.7.0_linux_amd64.tar.gz`) to load registry.
- For **air-gapped environment** ONLY:
    - Linux machine (bastion) that has access to the existing VPC.
    - The `dkp` binary on the bastion.
    - kubectl[830] for interacting with the running cluster on the bastion.
    - An existing local registry (see page 1606).
    - Ability to download artifacts from the internet and then copy those onto your bootstrap machine.
    - An AWS Air-Gapped Machine Image (see page 1637)

825 https://docs.docker.com/get-docker/
826 https://podman.io/getting-started/installation
827 https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
828 https://kubernetes.io/docs/tasks/tools/#kubectl
829 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html
830 https://kubernetes.io/docs/tasks/tools/#kubectl

On macOS, Docker runs in a virtual machine. Configure this virtual machine with at least 8GB of memory.

6.12.6.2.1 Control Plane Nodes

You should have at least three control plane nodes. Each control plane node should have at least:

- 4 cores
- 16 GiB memory
- Approximately 80 GiB of free space for the volume used for /var/lib/kubelet and /var/lib/containerd.
- Disk usage must be below 85% on the root volume.

DKP on AWS defaults to deploying an `m5.xlarge` instance with an 80GiB root volume for control plane nodes, which meets the above requirements.

6.12.6.2.2 Worker Nodes

You should have at least four worker nodes. The specific number of worker nodes required for your environment can vary depending on the cluster workload and size of the nodes. Each worker node should have at least:

- 8 cores
- 32 GiB memory
- Around 80 GiB of free space for the volume used for /var/lib/kubelet and /var/lib/containerd.
- Disk usage must be below 85% on the root volume.

DKP on AWS defaults to deploying a `m5.2xlarge` instance with an 80GiB root volume for worker nodes, which meets the above requirements.

If you use these instructions to create a cluster on AWS using the DKP default settings without any edits to configuration files or additional flags, your cluster is deployed on an Ubuntu 20.04 operating system image (see page 67) with 3 control plane nodes, and 4 worker nodes which match the requirements above.

Using these default images work, but due to missing optimizations, the created cluster will have certain limits. We suggest using Konvoy Image Builder to create a custom AMI[831] to take advantage of enhanced cluster operations.

831 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/143891417

6.12.6.3 2. AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

- Create AWS Minimal Permissions and Role to Create Clusters (see page 1156)
- Create AWS Cluster IAM Policies, Roles, and Artifacts (see page 1162)
- For multi-tenancy, every tenant should be in a different AWS account to ensure they are truly independent of other tenants in order to enforce security.
- Export the AWS region where you want to deploy the cluster:

```
export AWS_REGION=us-west-2
```

- Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

```
export AWS_PROFILE=<profile>
```

6.12.6.3.1 Next Step

AWS Using Konvoy Image Builder (see page 1153)

6.12.6.4 AWS Using Konvoy Image Builder

You must have at least one image before creating a new cluster. As long as you have an image, this step in your configuration is not required each time since that image can be used to spin up a new cluster. However, if you need different images for different environments or providers, you will need to create a new custom image. Inside the sections for either Non-air-gapped or Air-gapped install, you will find the instructions for how to create and apply custom images during cluster creation.

The following section describes how to use Konvoy Image Builder (KIB) with Amazon Web Services (AWS).

Konvoy Image Builder (see page 1626) has a more detailed section in the documentation if you need to refer there for compatible versions with DKP and other specific information.

6.12.6.4.1 Learn how to build a custom AMI for use with DKP

AMI images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create an AMI image of your current computer system settings and software. The AMI image can then be replicated and distributed, creating your computer system for other users. You can use overrides files to customize some of the components installed on your machine image. For example, you could tell KIB to install the FIPS versions of the Kubernetes components.

This procedure describes how to use the Konvoy Image Builder (see page 1626) (KIB) to create a Cluster API[832] compliant Amazon Machine Image (AMI). KIB uses variable overrides (see page 1670) to specify base image and container images to use in your new AMI.

> ⚠ In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1629). Explore the Customize your Image (see page 1661) topic for more options about overrides.

6.12.6.4.2 Prerequisites

Before you begin, you must:

- Download the KIB (see page 0) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Check the Supported Infrastructure Operating Systems (see page 67)
- Check the Supported Kubernetes Version (see page 1706) for your Provider.
- Create a working registry:
  - Version 4.0 of Podman[833]or higher for Linux. Host requirements found here: https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
  - Version 20.10.0 of Docker[834] or higher for Linux or MacOS
- Ensure you have met the minimal set of permissions from the AWS Image Builder Book[835].
- A Minimal IAM Permissions for KIB (see page 1629) to create an Image for an AWS account using Konvoy Image Builder.

6.12.6.4.2.1 Extract KIB Bundle

If not done previously during Konvoy Image Builder download in Prerequisites, extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION` folder. **Otherwise, proceed to Build the Image.**

In previous DKP releases, the distro package bundles were included in the downloaded air-gapped bundle. Currently, that air-gapped bundle contains the following artifacts with the exception of the distro packages:

- DKP Kubernetes packages
- Python packages (provided by upstream)
- Containerd tarball

832 https://cluster-api.sigs.k8s.io/
833 https://podman.io/getting-started/installation
834 https://www.docker.com/
835 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis

1. Download (see page 76) `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz` , and extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz && cd dkp-v2.8.0/kib
```

2. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.
3. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle` . This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml` . Example command:

```
./konvoy-image create-package-bundle --os redhat-8.4 --output-
directory=artifacts
```

NOTE: For FIPS, pass the flag: `--fips`

NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY` . Example command:

```
export RHSM_ACTIVATION_KEY="-ci"
export RHSM_ORG_ID="1232131"
```

4. Follow the instructions below to build an AMI.

The `konvoy-image` binary and all supporting folders are also extracted. When run, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

- Set environment variables for AWS access[836]. The following variables must be set using your credentials including required IAM (see page 1629):

```
export AWS_ACCESS_KEY_ID
export AWS_SECRET_ACCESS_KEY
export AWS_DEFAULT_REGION
```

- If you have an override file (see page 1670) to configure specific attributes of your AMI file, add it. Instructions for customizing an override file are found on this page: Image Overrides (see page 1678)

836 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-envvars.html

Inside the sections for either Non-air-gapped (see page 1190) or Air-gapped (see page 1218) cluster creation, you will find the instructions for how to apply custom images.

6.12.6.4.2.2 Next Step

AWS Minimal Permissions and Role to Create Clusters (see page 1156)

6.12.6.5 AWS Minimal Permissions and Role to Create Clusters

Configure IAM Prerequisites before starting a cluster

This section guides you in creating and using a minimally-scoped policy to create DKP clusters on an AWS account. For multi-tenancy, every tenant should be in a different AWS account to ensure they are truly independent of other tenants in order to enforce security.

6.12.6.5.1 Prerequisites

Before applying the IAM Policies, verify the following:

- You have a valid AWS account with credentials configured[837] that can manage CloudFormation Stacks, IAM Policies, IAM Roles, and IAM Instance Profiles.
- The AWS CLI utility[838] is installed.

6.12.6.5.2 Minimal Permissions

The following is an AWSCloudFormation stack that creates:

- A policy named `dkp-bootstrapper-policy` that enumerates the minimal permissions for a user that can create dkp aws clusters.
- A role named `dkp-bootstrapper-role` that uses the `dkp-bootstrapper-policy` with a trust policy to allow IAM users and ec2 instances from `MYAWSACCOUNTID` to use the role via STS.
- An instance profile `DKPBootstrapInstanceProfile` that wraps the `dkp-bootstrapper-role` to be used by ec2 instances.

6.12.6.5.3 Create Resources in CloudFormation Stack

To create the resources in the CloudFormation stack:

1. Copy the following contents into a file:

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
```

837 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html
838 https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html

```
AWSIAMInstanceProfileDKPBootstrapper:
  Properties:
    InstanceProfileName: DKPBootstrapInstanceProfile
    Roles:
    - Ref: DKPBootstrapRole
  Type: AWS::IAM::InstanceProfile
AWSIAMManagedPolicyDKPBootstrapper:
  Properties:
    Description: Minimal policy to create dkp clusters in AWS
    ManagedPolicyName: dkp-bootstrapper-policy
    PolicyDocument:
      Statement:
      - Action:
        - ec2:AllocateAddress
        - ec2:AssociateRouteTable
        - ec2:AttachInternetGateway
        - ec2:AuthorizeSecurityGroupIngress
        - ec2:CreateInternetGateway
        - ec2:CreateNatGateway
        - ec2:CreateRoute
        - ec2:CreateRouteTable
        - ec2:CreateSecurityGroup
        - ec2:CreateSubnet
        - ec2:CreateTags
        - ec2:CreateVpc
        - ec2:ModifyVpcAttribute
        - ec2:DeleteInternetGateway
        - ec2:DeleteNatGateway
        - ec2:DeleteRouteTable
        - ec2:DeleteSecurityGroup
        - ec2:DeleteSubnet
        - ec2:DeleteTags
        - ec2:DeleteVpc
        - ec2:DescribeAccountAttributes
        - ec2:DescribeAddresses
        - ec2:DescribeAvailabilityZones
        - ec2:DescribeInstanceTypes
        - ec2:DescribeInternetGateways
        - ec2:DescribeImages
        - ec2:DescribeNatGateways
        - ec2:DescribeNetworkInterfaces
        - ec2:DescribeNetworkInterfaceAttribute
        - ec2:DescribeRouteTables
        - ec2:DescribeSecurityGroups
        - ec2:DescribeSubnets
        - ec2:DescribeVpcs
        - ec2:DescribeVpcAttribute
        - ec2:DescribeVolumes
        - ec2:DetachInternetGateway
        - ec2:DisassociateRouteTable
        - ec2:DisassociateAddress
        - ec2:ModifyInstanceAttribute
```

```
    - ec2:ModifyInstanceMetadataOptions
    - ec2:ModifyNetworkInterfaceAttribute
    - ec2:ModifySubnetAttribute
    - ec2:ReleaseAddress
    - ec2:RevokeSecurityGroupIngress
    - ec2:RunInstances
    - ec2:TerminateInstances
    - tag:GetResources
    - elasticloadbalancing:AddTags
    - elasticloadbalancing:CreateLoadBalancer
    - elasticloadbalancing:ConfigureHealthCheck
    - elasticloadbalancing:DeleteLoadBalancer
    - elasticloadbalancing:DescribeLoadBalancers
    - elasticloadbalancing:DescribeLoadBalancerAttributes
    - elasticloadbalancing:DescribeTargetGroups
    - elasticloadbalancing:ApplySecurityGroupsToLoadBalancer
    - elasticloadbalancing:DescribeTags
    - elasticloadbalancing:ModifyLoadBalancerAttributes
    - elasticloadbalancing:RegisterInstancesWithLoadBalancer
    - elasticloadbalancing:DeregisterInstancesFromLoadBalancer
    - elasticloadbalancing:RemoveTags
    - autoscaling:DescribeAutoScalingGroups
    - autoscaling:DescribeInstanceRefreshes
    - ec2:CreateLaunchTemplate
    - ec2:CreateLaunchTemplateVersion
    - ec2:DescribeLaunchTemplates
    - ec2:DescribeLaunchTemplateVersions
    - ec2:DeleteLaunchTemplate
    - ec2:DeleteLaunchTemplateVersions
    - ec2:DescribeKeyPairs
    Effect: Allow
    Resource:
    - '*'
  - Action:
    - autoscaling:CreateAutoScalingGroup
    - autoscaling:UpdateAutoScalingGroup
    - autoscaling:CreateOrUpdateTags
    - autoscaling:StartInstanceRefresh
    - autoscaling:DeleteAutoScalingGroup
    - autoscaling:DeleteTags
    Effect: Allow
    Resource:
    - arn:*:autoscaling:*:*:autoScalingGroup:*:autoScalingGroupName/*
  - Action:
    - ecr:DescribeRepositories
    - ecr:CreateRepository
    - ecr:PutLifecyclePolicy
    - ecr:CompleteLayerUpload
    - ecr:GetAuthorizationToken
    - ecr:UploadLayerPart
    - ecr:InitiateLayerUpload
    - ecr:BatchCheckLayerAvailability
```

```
          - ecr:BatchGetImage
          - ecr:GetDownloadUrlForLayer
          - ecr:PutImage
          Effect: Allow
          Resource:
          - arn:aws:ecr:*:MYAWSACCOUNT:repository/*
        - Action:
          - iam:CreateServiceLinkedRole
          Condition:
            StringLike:
              iam:AWSServiceName: autoscaling.amazonaws.com
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/aws-service-role/autoscaling.amazonaws.com/
AWSServiceRoleForAutoScaling
        - Action:
          - iam:CreateServiceLinkedRole
          Condition:
            StringLike:
              iam:AWSServiceName: elasticloadbalancing.amazonaws.com
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/aws-service-role/
elasticloadbalancing.amazonaws.com/AWSServiceRoleForElasticLoadBalancing
        - Action:
          - iam:CreateServiceLinkedRole
          Condition:
            StringLike:
              iam:AWSServiceName: spot.amazonaws.com
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/aws-service-role/spot.amazonaws.com/
AWSServiceRoleForEC2Spot
        - Action:
          - iam:PassRole
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/*.cluster-api-provider-aws.sigs.k8s.io
        - Action:
          - secretsmanager:CreateSecret
          - secretsmanager:DeleteSecret
          - secretsmanager:TagResource
          Effect: Allow
          Resource:
          - arn:*:secretsmanager:*:*:secret:aws.cluster.x-k8s.io/*
        Version: 2012-10-17
      Roles:
      - Ref: DKPBootstrapRole
    Type: AWS::IAM::ManagedPolicy
  DKPBootstrapRole:
    Properties:
      AssumeRolePolicyDocument:
```

```
        Statement:
        - Action:
          - sts:AssumeRole
          Effect: Allow
          Principal:
            Service:
            - ec2.amazonaws.com
        - Action:
            - sts:AssumeRole
          Effect: Allow
          Principal:
                AWS: arn:aws:iam::MYAWSACCOUNT:root
        Version: 2012-10-17
      RoleName: dkp-bootstrapper-role
    Type: AWS::IAM::Role
```

If your organization uses Flatcar, add the following s3 permissions to your CloudFormation stack in the `dkp-bootstrapper-policy` :

```
- Action:
          - 's3:CreateBucket'
          - 's3:DeleteBucket'
          - 's3:PutObject'
          - 's3:DeleteObject'
          - 's3:PutBucketPolicy'
          Effect: Allow
          Resource:
          - 'arn:*:s3:::cluster-api-provider-aws-*'
```

2. Replace the following with the correct values:
   a. `MYFILENAME.yaml` - give your file a meaningful name.
   b. `MYSTACKNAME` - give your cloudformation stack a meaningful name.
   c. `MYAWSACCOUNT` - replace with an AWS Account ID number such as: `111122223333`
3. Run the following command to create the stack :

```
aws cloudformation create-stack --template-body=file://MYFILENAME.yaml --stack-
name=MYSTACKNAME --capabilities CAPABILITY_NAMED_IAM
```

6.12.6.5.4 Leverage the Role

Use temporary User Access Keys via STS.

The created `dkp-bootstrapper-role` can be assumed by IAM users for temporary credentials via STS by running the command below:

```
aws sts assume-role --role-arn arn:aws:iam::MYAWSACCOUNT:role/dkp-bootstrapper-role
--role-session-name EXAMPLE
```

Which returns something similar to this:

```
{
    "Credentials": {
        "AccessKeyId": "ASIA6RTF53ZH5B52EVM5",
        "SecretAccessKey": "BSssyvSsdfJY74jubsadfdsafdsaH7x1L+8Vk/",
        "SessionToken": "IQoJb3JpZ2z5cyChb9PtJvP0S6KAi",
        "Expiration": "2022-07-14T20:19:13+00:00"
    },
    "AssumedRoleUser": {
        "AssumedRoleId": "ASIA6RTF53ZH5B52EVM5:test",
        "Arn": "arn:aws:sts::MYAWSACCOUNTID:assumed-role/dkp-bootstrapper-role/test"
    }
}
```

And then `export` the following environment variables with the results:

```
export AWS_ACCESS_KEY_ID=(.Credentials.AccessKeyId)

export AWS_SECRET_ACCESS_KEY=(.Credentials.SecretAccessKey)

export AWS_SESSION_TOKEN=(.Credentials.SessionToken)
```

These credentials are short lived and would need to be updated in the bootstrap cluster

6.12.6.5.5 Use EC2 Instance Profiles

The created `dkp-bootstrapper-role` can be assumed by an ec2 instance a user would run `dkp create cluster` commands from. To do this, specify the IAM Instance Profile `DKPBootstrapInstanceProfile` on creation.

6.12.6.5.6 Use Access Keys

AWS administrators can attach the `dkp-bootstrapper-policy` to an existing IAM user[839] and authenticate with Access Keys[840] on the work station they would run `dkp create cluster` commands from by exporting the following environment variables with the appropriate values for the IAM user.

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_DEFAULT_REGION=us-west-2
```

In regards to Access Keys usage, a system administrator should always consider AWS's Best practices[841].

> EKS cluster minimal permissions are required if attaching an EKS cluster. Refer to Minimal User Permission for EKS Cluster Creation (see page 1263). The CloudFormation stack on that page adds a policy named `eks-bootstrapper` to manage EKS cluster to the `dkp-bootstrapper-role` created by the CloudFormation stack on this page.

If your organization uses encrypted AMI's (https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIEncryption.html), then you will need to add additional permissions to the control plane policy to allow access to the Amazon Key Management Services. See the following documentation for information on the necessary policies you may need: AWS Key Policies[842].

Return to EKS Cluster IAM Permissions and Roles (see page 1266) or proceed to the next AWS step below.

6.12.6.5.7 Next Step:

AWS Cluster IAM Policies, Roles, and Artifacts (see page 1162)

6.12.6.6 AWS Cluster IAM Policies, Roles, and Artifacts

This guides a DKP user in creating IAM Policies and Instance Profiles used by the cluster's control plane and worker nodes using the provided AWS CloudFormation Stack. For multi-tenancy, every tenant should be in a different AWS account to ensure they are truly independent of other tenants in order to enforce security.

6.12.6.6.1 Prerequisites

Before applying the IAM Policies, verify the following:

839 https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_manage-attach-detach.html
840 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html
841 https://docs.aws.amazon.com/general/latest/gr/aws-access-keys-best-practices.html
842 https://docs.aws.amazon.com/kms/latest/developerguide/key-policy-default.html#key-policy-service-integration

- You have a valid AWS account with credentials configured[843] that can manage CloudFormation Stacks, IAM Policies, IAM Roles, and IAM Instance Profiles.
- You have the AWS CLI utility installed[844].

Below is information regarding the set up for policies and roles. After reading the information for each of these areas, you will find the CloudFormation Stack that creates these policies and roles in the IAM Artifacts drop-down section below:

Policies

1. `AWSIAMManagedPolicyCloudProviderControlPlane` enumerates the Actions required by the workload cluster control plane machines. It is attached to the `AWSIAMRoleControlPlane` Role.
2. `AWSIAMManagedPolicyCloudProviderNodes` enumerates the Actions required by the workload cluster worker machines. It is attached to the `AWSIAMRoleNodes` Role.
3. `AWSIAMManagedPolicyControllers` enumerates the Actions required by the workload cluster worker machines. It is attached to the `AWSIAMRoleControlPlane` Role.

Roles

1. `AWSIAMRoleControlPlane` is the Role associated with the `AWSIAMInstanceProfileControlPlane` Instance Profile.
2. `AWSIAMRoleNodes` is the Role associated with the `AWSIAMInstanceProfileNodes` Instance Profile.

For more information on learning how to grant cluster access to IAM users and roles, see the official AWS Documentation[845].

IAM Artifacts

6.12.6.6.2 Instance Profiles

1. `AWSIAMInstanceProfileControlPlane`, assigned to workload cluster control plane machines.

> If the name is changed from the default, used below, it must be passed to `dkp create cluster` with the `--control-plane-iam-instance-profile` flag.

2. `AWSIAMInstanceProfileNodes`, assigned to workload cluster worker machines.

843 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html
844 https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html
845 https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html

If the name is changed from the default, used below, it must be passed to `dkp create cluster` with the `--worker-iam-instance-profile` flag.

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  AWSIAMInstanceProfileControlPlane:
    Properties:
      InstanceProfileName: control-plane.cluster-api-provider-aws.sigs.k8s.io
      Roles:
      - Ref: AWSIAMRoleControlPlane
    Type: AWS::IAM::InstanceProfile
  AWSIAMInstanceProfileNodes:
    Properties:
      InstanceProfileName: nodes.cluster-api-provider-aws.sigs.k8s.io
      Roles:
      - Ref: AWSIAMRoleNodes
    Type: AWS::IAM::InstanceProfile
  AWSIAMManagedPolicyCloudProviderControlPlane:
    Properties:
      Description: For the Kubernetes Cloud Provider AWS Control Plane
      ManagedPolicyName: control-plane.cluster-api-provider-aws.sigs.k8s.io
      PolicyDocument:
        Statement:
        - Action:
          - autoscaling:DescribeAutoScalingGroups
          - autoscaling:DescribeLaunchConfigurations
          - autoscaling:DescribeTags
          - ec2:DescribeInstances
          - ec2:DescribeImages
          - ec2:DescribeRegions
          - ec2:DescribeRouteTables
          - ec2:DescribeSecurityGroups
          - ec2:DescribeSubnets
          - ec2:DescribeVolumes
          - ec2:CreateSecurityGroup
          - ec2:CreateTags
          - ec2:CreateVolume
          - ec2:ModifyInstanceAttribute
          - ec2:ModifyVolume
          - ec2:AttachVolume
          - ec2:AuthorizeSecurityGroupIngress
          - ec2:CreateRoute
          - ec2:DeleteRoute
          - ec2:DeleteSecurityGroup
          - ec2:DeleteVolume
          - ec2:DetachVolume
          - ec2:RevokeSecurityGroupIngress
```

```
        - ec2:DescribeVpcs
        - elasticloadbalancing:AddTags
        - elasticloadbalancing:AttachLoadBalancerToSubnets
        - elasticloadbalancing:ApplySecurityGroupsToLoadBalancer
        - elasticloadbalancing:CreateLoadBalancer
        - elasticloadbalancing:CreateLoadBalancerPolicy
        - elasticloadbalancing:CreateLoadBalancerListeners
        - elasticloadbalancing:ConfigureHealthCheck
        - elasticloadbalancing:DeleteLoadBalancer
        - elasticloadbalancing:DeleteLoadBalancerListeners
        - elasticloadbalancing:DescribeLoadBalancers
        - elasticloadbalancing:DescribeLoadBalancerAttributes
        - elasticloadbalancing:DetachLoadBalancerFromSubnets
        - elasticloadbalancing:DeregisterInstancesFromLoadBalancer
        - elasticloadbalancing:ModifyLoadBalancerAttributes
        - elasticloadbalancing:RegisterInstancesWithLoadBalancer
        - elasticloadbalancing:SetLoadBalancerPoliciesForBackendServer
        - elasticloadbalancing:AddTags
        - elasticloadbalancing:CreateListener
        - elasticloadbalancing:CreateTargetGroup
        - elasticloadbalancing:DeleteListener
        - elasticloadbalancing:DeleteTargetGroup
        - elasticloadbalancing:DescribeListeners
        - elasticloadbalancing:DescribeLoadBalancerPolicies
        - elasticloadbalancing:DescribeTargetGroups
        - elasticloadbalancing:DescribeTargetHealth
        - elasticloadbalancing:ModifyListener
        - elasticloadbalancing:ModifyTargetGroup
        - elasticloadbalancing:RegisterTargets
        - elasticloadbalancing:SetLoadBalancerPoliciesOfListener
        - iam:CreateServiceLinkedRole
        - kms:DescribeKey
        - kms:CreateGrant
        Effect: Allow
        Resource:
        - '*'
      Version: 2012-10-17
    Roles:
    - Ref: AWSIAMRoleControlPlane
  Type: AWS::IAM::ManagedPolicy
AWSIAMManagedPolicyCloudProviderNodes:
  Properties:
    Description: For the Kubernetes Cloud Provider AWS nodes
    ManagedPolicyName: nodes.cluster-api-provider-aws.sigs.k8s.io
    PolicyDocument:
      Statement:
      - Action:
        - ec2:DescribeInstances
        - ec2:DescribeRegions
        - ecr:GetAuthorizationToken
        - ecr:BatchCheckLayerAvailability
        - ecr:GetDownloadUrlForLayer
```

```
        - ecr:GetRepositoryPolicy
        - ecr:DescribeRepositories
        - ecr:ListImages
        - ecr:BatchGetImage
        Effect: Allow
        Resource:
        - '*'
      - Action:
        - secretsmanager:DeleteSecret
        - secretsmanager:GetSecretValue
        Effect: Allow
        Resource:
        - arn:*:secretsmanager:*:*:secret:aws.cluster.x-k8s.io/*
      - Action:
        - ssm:UpdateInstanceInformation
        - ssmmessages:CreateControlChannel
        - ssmmessages:CreateDataChannel
        - ssmmessages:OpenControlChannel
        - ssmmessages:OpenDataChannel
        - s3:GetEncryptionConfiguration
        Effect: Allow
        Resource:
        - '*'
      Version: 2012-10-17
    Roles:
    - Ref: AWSIAMRoleControlPlane
    - Ref: AWSIAMRoleNodes
  Type: AWS::IAM::ManagedPolicy
AWSIAMManagedPolicyControllers:
  Properties:
    Description: For the Kubernetes Cluster API Provider AWS Controllers
    ManagedPolicyName: controllers.cluster-api-provider-aws.sigs.k8s.io
    PolicyDocument:
      Statement:
      - Action:
        - ec2:AllocateAddress
        - ec2:AssociateRouteTable
        - ec2:AttachInternetGateway
        - ec2:AuthorizeSecurityGroupIngress
        - ec2:CreateInternetGateway
        - ec2:CreateNatGateway
        - ec2:CreateRoute
        - ec2:CreateRouteTable
        - ec2:CreateSecurityGroup
        - ec2:CreateSubnet
        - ec2:CreateTags
        - ec2:CreateVpc
        - ec2:ModifyVpcAttribute
        - ec2:DeleteInternetGateway
        - ec2:DeleteNatGateway
        - ec2:DeleteRouteTable
        - ec2:DeleteSecurityGroup
```

```
      - ec2:DeleteSubnet
      - ec2:DeleteTags
      - ec2:DeleteVpc
      - ec2:DescribeAccountAttributes
      - ec2:DescribeAddresses
      - ec2:DescribeAvailabilityZones
      - ec2:DescribeInstanceTypes
      - ec2:DescribeInternetGateways
      - ec2:DescribeImages
      - ec2:DescribeNatGateways
      - ec2:DescribeNetworkInterfaces
      - ec2:DescribeNetworkInterfaceAttribute
      - ec2:DescribeRouteTables
      - ec2:DescribeSecurityGroups
      - ec2:DescribeSubnets
      - ec2:DescribeVpcs
      - ec2:DescribeVpcAttribute
      - ec2:DescribeVolumes
      - ec2:DetachInternetGateway
      - ec2:DisassociateRouteTable
      - ec2:DisassociateAddress
      - ec2:ModifyInstanceAttribute
      - ec2:ModifyInstanceMetadataOptions
      - ec2:ModifyNetworkInterfaceAttribute
      - ec2:ModifySubnetAttribute
      - ec2:ReleaseAddress
      - ec2:RevokeSecurityGroupIngress
      - ec2:RunInstances
      - ec2:TerminateInstances
      - tag:GetResources
      - elasticloadbalancing:AddTags
      - elasticloadbalancing:CreateLoadBalancer
      - elasticloadbalancing:ConfigureHealthCheck
      - elasticloadbalancing:DeleteLoadBalancer
      - elasticloadbalancing:DescribeLoadBalancers
      - elasticloadbalancing:DescribeLoadBalancerAttributes
      - elasticloadbalancing:DescribeTargetGroups
      - elasticloadbalancing:ApplySecurityGroupsToLoadBalancer
      - elasticloadbalancing:DescribeTags
      - elasticloadbalancing:ModifyLoadBalancerAttributes
      - elasticloadbalancing:RegisterInstancesWithLoadBalancer
      - elasticloadbalancing:DeregisterInstancesFromLoadBalancer
      - elasticloadbalancing:RemoveTags
      - autoscaling:DescribeAutoScalingGroups
      - autoscaling:DescribeInstanceRefreshes
      - ec2:CreateLaunchTemplate
      - ec2:CreateLaunchTemplateVersion
      - ec2:DescribeLaunchTemplates
      - ec2:DescribeLaunchTemplateVersions
      - ec2:DeleteLaunchTemplate
      - ec2:DeleteLaunchTemplateVersions
      Effect: Allow
```

```
          Resource:
          - '*'
        - Action:
          - autoscaling:CreateAutoScalingGroup
          - autoscaling:UpdateAutoScalingGroup
          - autoscaling:CreateOrUpdateTags
          - autoscaling:StartInstanceRefresh
          - autoscaling:DeleteAutoScalingGroup
          - autoscaling:DeleteTags
          Effect: Allow
          Resource:
          - arn:*:autoscaling:*:*:autoScalingGroup:*:autoScalingGroupName/*
        - Action:
          - iam:CreateServiceLinkedRole
          Condition:
            StringLike:
              iam:AWSServiceName: autoscaling.amazonaws.com
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/aws-service-role/autoscaling.amazonaws.com/
AWSServiceRoleForAutoScaling
        - Action:
          - iam:CreateServiceLinkedRole
          Condition:
            StringLike:
              iam:AWSServiceName: elasticloadbalancing.amazonaws.com
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/aws-service-role/elasticloadbalancing.amazonaws.com/
AWSServiceRoleForElasticLoadBalancing
        - Action:
          - iam:CreateServiceLinkedRole
          Condition:
            StringLike:
              iam:AWSServiceName: spot.amazonaws.com
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/aws-service-role/spot.amazonaws.com/
AWSServiceRoleForEC2Spot
        - Action:
          - iam:PassRole
          Effect: Allow
          Resource:
          - arn:*:iam::*:role/*.cluster-api-provider-aws.sigs.k8s.io
        - Action:
          - secretsmanager:CreateSecret
          - secretsmanager:DeleteSecret
          - secretsmanager:TagResource
          Effect: Allow
          Resource:
          - arn:*:secretsmanager:*:*:secret:aws.cluster.x-k8s.io/*
        Version: 2012-10-17
```

```
      Roles:
      - Ref: AWSIAMRoleControlPlane
    Type: AWS::IAM::ManagedPolicy
  AWSIAMRoleControlPlane:
    Properties:
      AssumeRolePolicyDocument:
        Statement:
        - Action:
          - sts:AssumeRole
          Effect: Allow
          Principal:
            Service:
            - ec2.amazonaws.com
        Version: 2012-10-17
      RoleName: control-plane.cluster-api-provider-aws.sigs.k8s.io
    Type: AWS::IAM::Role
  AWSIAMRoleNodes:
    Properties:
      AssumeRolePolicyDocument:
        Statement:
        - Action:
          - sts:AssumeRole
          Effect: Allow
          Principal:
            Service:
            - ec2.amazonaws.com
        Version: 2012-10-17
      RoleName: nodes.cluster-api-provider-aws.sigs.k8s.io
    Type: AWS::IAM::Role
```

To create the resources in the cloudformation stack copy the contents above into a file replacing `MYFILENAME.yaml` and `MYSTACKNAME` with the intended values. Then execute the following command:

```
 aws cloudformation create-stack --template-body=file://MYFILENAME.yaml --stack-
name=MYSTACKNAME --capabilities  CAPABILITY_NAMED_IAM
```

If your organization uses encrypted AMIs[846], then you will need to add additional permissions to the control plane policy `control-plane.cluster-api-provider-aws.sigs.k8s.io` to allow access to the Amazon Key Management Services. The code snippet shows how to add a particular key ARN that is used to encrypt and decrypt AMIs.

```
---
Action:
```

846 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIEncryption.html

```
- kms:CreateGrant
- kms:DescribeKey
- kms:Encrypt
- kms:Decrypt
- kms:ReEncrypt*
- kms:GenerateDataKey*
Resource:
- arn:aws:kms:us-west-2:111122223333:key/key-arn
Effect: Allow
```

If your organization uses Flatcar, then you will need to add additional permissions to the control plane policy `control-plane.cluster-api-provider-aws.sigs.k8s.io`

```
PolicyDocument:
  Statement:
  ...
  - Action:
    - 's3:CreateBucket'
    - 's3:DeleteBucket'
    - 's3:PutObject'
    - 's3:DeleteObject'
    - 's3:PutBucketPolicy'
    - 's3:PutBucketTagging'
    Effect: Allow
    Resource:
    - 'arn:*:s3:::cluster-api-provider-aws-*'
```

EKS IAM Policies and Instance Profiles that govern who has access to the cluster are found here: EKS Cluster IAM Permissions and Roles (see page 1266). If attaching an EKS cluster, that CloudStack Formation will also need to be run to create the ARN of the bootstrapper role.

Return to EKS - EKS Cluster IAM Permissions and Roles (see page 1266) or proceed to the next AWS step below.

6.12.6.6.3 Next Steps:

AWS Cluster Creation Customization Choices (see page 1170)

6.12.7 AWS Cluster Creation Customization Choices

Below are a some methods to customize your cluster during creation. If none of these choices apply, skip this page and proceed to either section:

- AWS Install in a Non-air-gapped Environment (see page 1184)
- AWS Install in an Air-gapped Environment (see page 1207)

6.12.7.1 Section Topics

When creating clusters, many options are available such as those listed in this section of the documentation. Familiarize yourself with the flags required to apply these customizations during cluster creation.

- AWS Naming Cluster (see page 1172) — In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.
- AWS Customizing CAPI Clusters (see page 1173) — Familiarize yourself with Cluster API before editing the cluster objects because edits can prevent the cluster from deploying successfully.
- AWS Custom AMI (see page 1173) — To use a custom AMI when creating your cluster, you must create that AMI using Konvoy Image Builder(KIB) first.
- AWS Registry Mirrors (see page 1174) — In an air-gapped environment, you need a local repository to store Helm charts, Docker images, and other artifacts. In an environment with access to the Internet, you can retrieve artifacts from specialized repositories dedicated to them, such as Docker images contained in DockerHub and Helm Charts that come from a dedicated Helm Chart repository.
- AWS Loading the Registry (see page 1175) — Because air-gapped environments do not have direct access to the Internet, you must download, extract and load several required images to your local container registry, before installing DKP. Refer to Registry Mirror Tools for specific information.
- AWS Registry Configuration (see page 1180) — Configure your cluster to use an existing local registry when attempting to pull images by adding the flag(s) to the dkp create cluster command to pull images from your local registry.
- AWS HTTP Proxy (see page 1182) — When creating a DKP cluster in environments that use an HTTP/HTTPS proxy, you must provide proxy details. The proxy values are strings that list a set of proxy servers, URLs, or wildcard addresses that is specific to your environment.
- AWS Output Directory YAML (see page 1183) — You can create individual files with different smaller manifests for ease in editing using the --output-directory flag used with --output=json|yaml. You create the directory of where to output resources to files.
- AWS Provision on the Flatcar Linux OS (see page 1183) — Flatcar default network interface name might require specifying. It is most likely to be ens192 requiring passing the parameter --virtual-ip-interface ens192 to the dkp create cluster aws command. Otherwise, the cluster creation might fail because kube-vip can not configure the first control-plane virtual IP.

6.12.7.1.1 Next Step

After you make decisions about customization choices, proceed to either the non-air-gapped or air-gapped environment instructions:

- AWS Install in a Non-air-gapped Environment (see page 1184)
- AWS Install in an Air-gapped Environment (see page 1207)

6.12.7.2 AWS Naming Cluster

6.12.7.2.1 Name Your Cluster

In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name.

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[847] for more naming information.

Follow these steps:

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<aws-example>
```

6.12.7.2.1.1 Cluster Name List in AWS

To get a list of names used in your AWS account, use the `aws` CLI[848]. After downloading, use the following command:

```
aws ec2 describe-vpcs --filter "Name=tag-key,Values=kubernetes.io/cluster" --query
"Vpcs[*].Tags[?Key=='kubernetes.io/cluster'].Value | sort(@[*][0])"
```

```
    "alex-aws-cluster-afe98",
    "sam-aws-cluster-8if9q"
```

- (Optional) To create a cluster name that is unique, use the following command:

```
export CLUSTER_NAME=aws-example-$(LC_CTYPE=C tr -dc 'a-z0-9' </dev/urandom |
fold -w 5 | head -n1)
echo $CLUSTER_NAME
```

847 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/
848 https://aws.amazon.com/cli/

```
aws-example-pf4a3
```

This will create a unique name every time you run it, so use it with forethought.

6.12.7.2.1.2 Next Topic

AWS Customizing CAPI Clusters (see page 1173)

6.12.7.3 AWS Customizing CAPI Clusters

Familiarize yourself with Cluster API (see page 1061) before editing the cluster objects because edits can prevent the cluster from deploying successfully.

The result of this command will allow such edits:

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--dry-run \
--output=yaml \
> ${CLUSTER_NAME}.yaml
```

To edit the YAML, you need to understand the CAPI components to avoid breaking the cluster. For expanded component explanation, refer to the Universal Configurations for All Providers section of the documentation in the section: Customizing CAPI Components for a Cluster (see page 1061)

6.12.7.3.1 Next Topic

AWS Custom AMI (see page 1173)

6.12.7.4 AWS Custom AMI

To use a custom AMI when creating your cluster, you must create that AMI using Konvoy Image Builder (see page 1626)(KIB) first.

Then perform the export and name the custom AMI for use in the command `dkp create cluster` after this step:

```
export AWS_AMI_ID=ami-<ami-id-here>
```

IMPORTANT: The AMI must be created with Konvoy Image Builder (see page 1626) in order to use the registry mirror feature.

Set the environment variable for the AMI you choose during the `dkp create cluster` command. This will output the generated manifest into a new file as shown in the example command:

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--ami=${AWS_AMI_ID} \
--dry-run \
--output=yaml \
> ${CLUSTER_NAME}.yaml
```

This step will be performed during both AWS Install in a Non-air-gapped Environment (see page 1184) and AWS Install in an Air-gapped Environment (see page 1207) and explained prior to cluster creation on a page in both of those scenarios:

- Custom AMI in Cluster Creation (see page 1185)
- Custom AMI in Air-gapped Cluster Creation (see page 1208)

6.12.7.4.1 Next Topic

AWS Registry Mirrors (see page 1174)

6.12.7.5 AWS Registry Mirrors

In an air-gapped environment, you need a local repository to store Helm charts, Docker images, and other artifacts. In an environment with access to the Internet, you can retrieve artifacts from specialized repositories dedicated to them, such as Docker images contained in DockerHub and Helm Charts that come from a dedicated Helm Chart repository.

Kubernetes does not natively provide a registry for hosting the container images you will use to run the applications you want to deploy on Kubernetes. Instead, Kubernetes requires you to use an external solution for storing and sharing container images. There are a variety of Kubernetes-compatible registry options that are compatible with DKP.

6.12.7.5.1 How Does it Work?

The **first time** you request an image from your local registry mirror, it pulls the image from the public registry (such as Docker) and stores it locally before handing it back to you. On subsequent requests, the local registry mirror is able to serve the image from its own storage.

6.12.7.5.2 Air-gapped vs Non-air-gapped Environments

In a non-air-gapped environment, you have access to the Internet. You retrieve artifacts from specialized repositories dedicated to them, such as Docker images contained in DockerHub and Helm Charts that come from a dedicated Helm Chart repository. You can also create your own local repository to hold the downloaded container images needed or any custom images you've created with the Konvoy Image Builder (see page 1626) tool.

In an **air-gapped environment**, you need a local repository to store Helm charts, Docker images, and other artifacts. Private registries provide security and privacy into enterprise container image storage, whether hosted remotely or on-premises locally in an air-gapped environment. DKP in an air-gapped environment

requires a local container registry of trusted images to enable production-level Kubernetes cluster management. However, a local registry is an option in a non-air-gapped environment as well for speed and security.

If you want to use images from this local registry to deploy applications inside your Kubernetes cluster, you'll need to set up a **secret** for a private registry. The secret contains your login data, which Kubernetes needs to connect to your private repository.

6.12.7.5.2.1 Related Topics

More information and detail can be found:

- Registry Mirror Tools (see page 1606)
- Use a Registry Mirror (see page 1609)

6.12.7.5.2.2 Next Topic

AWS Loading the Registry (see page 1175)

If none of the customizations apply, continue to installation instructions for your environment:

- AWS Install in a Non-air-gapped Environment (see page 1184)
  OR
- AWS Install in an Air-gapped Environment (see page 1207)

6.12.7.6 AWS Loading the Registry

Because air-gapped environments do not have direct access to the Internet, you must download, extract and load several required images to your local container registry, before installing DKP. Refer to Registry Mirror Tools (see page 1606) for specific information.

6.12.7.6.1 Load Images into your Registry

After you create an image for your Air-gapped environment, you will need to load your registry. However, local registries (see page 1606) can also be used in Non-air-gapped environments for speed and security reasons. If you choose to use this feature in your Non-air-gapped environment, then this page will apply to you.

6.12.7.6.2 Download all Images for Air-gapped Deployments

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for prerequisites to download and then how to push the necessary images to this registry.

1. Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`) to load registry images as explained below.
2. Connectivity with clusters attaching to the management cluster is required:

- Both management and attached clusters must be able to connect to the local registry.
- The management cluster must be able to connect to all attached cluster's API servers.
- The management cluster must be able to connect to any load balancers created for platform services on the management cluster.

6.12.7.6.3 Extract Air-gapped Images and Set Variables

Follow these steps to **extract** the air-gapped image bundles into your private registry using these examples for ECR:

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. For the bootstrap, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

   ```
   cd dkp-v2.8.1
   ```

3. Set an environment variable with your registry address for **ECR**:

   ```
   export REGISTRY_URL=<ecr-registry-URI>
   ```

NOTE: To use ECR:

```
export REGISTRY_URL=<ecr-registry-URI>
```

- `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- The environment where you are running the `dkp push` command must be authenticated with AWS in order to load your images into ECR.

Registries other than ECR

For other registries, more environment variables would be:

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
export REGISTRY_CA=<path to the cacert file on the bastion>
```

6.12.7.6.3.1 Load Images to your Private Registry - Konvoy

Before creating or upgrading a Kubernetes cluster, you need to load the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, refer to Local Registry Tools (see page 1606) page for more information.

Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

For specific `push` flags, refer to the dkp push bundle (see page 1910)section of CLI commands.

For registries other than ECR

If not using ECR, the `push` command will be different depending on username and password requirements:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

6.12.7.6.3.2 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL}
```

For registries other than ECR

If not using ECR, the `push` command will be different depending on username and password requirements:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

6.12.7.6.3.3 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL}
```

For registries other than ECR

If not using ECR, the `push` command will be different depending on username and password requirements:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME}
--to-registry-password=${REGISTRY_PASSWORD}
```

6.12.7.6.3.4 Related Topic

AWS Elastic Container Registry (ECR) (see page 1178)

6.12.7.6.3.5 Next Step

AWS Registry Configuration (see page 1180)

6.12.7.6.4 AWS Elastic Container Registry (ECR)

6.12.7.6.4.1 Load Images into your Registry

Because air-gapped environments do not have direct access to the Internet, you must download, extract and load several required images to your local container registry (see page 1606), before installing DKP.

This page is to explain ECR specifics, but assumes you have already downloaded (see page 76) and extracted (see page 1211) the bundle from the Prerequisites (see page 1151). The sections below explain further how you are pushing the images to your AWS ECR registry and then using them in creating a cluster.

AWS ECR

AWS ECR (Elastic Container Registry) is supported as your air-gapped image registry or a non-air-gapped registry mirror. DKP added support for using AWS ECR as a default registry when uploading image bundles in AWS.

Prerequisites

- Ensure you have followed the steps to create proper permissions in AWS Minimal Permissions and Role to Create Clusters (see page 1156)
- Ensure you have created AWS Cluster IAM Policies, Roles, and Artifacts (see page 1162)

Upload the Air-gapped Image Bundle to the Local ECR Registry:

A cluster administrator uses DKP CLI commands to upload the image bundle to ECR with parameters:

```
dkp push bundle --bundle <bundle> --to-registry=<ecr-registry-address>/<ecr-registry-name>
```

Parameter definitions:

- `--bundle <bundle>` the group of images. The example below is for the DKP air-gapped environment bundle
- `--to-registry=<ecr-registry-address>/<ecr-registry-name>` to provide registry location for push

An **example** command would be:

```
dkp push bundle --bundle container-images/konvoy-image-bundle-v2.8.1.tar --to-registry=333000009999.dkr.ecr.us-west-2.amazonaws.com/can-test
```

NOTE: You can aslo set an environment variable with your registry address for **ECR**:

```
export REGISTRY_URL=<ecr-registry-URI>
```

- `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- The environment where you are running the `dkp push` command must be authenticated with AWS in order to load your images into ECR.

Air-gapped Environment Information regarding your AWS ECR Account

The cluster administrator uses existing DKP CLI commands to create the cluster and refer to their internal ECR for image repository. The administrator does not need to provide static ECR registry credentials. See Use a Registry Mirror (see page 1609) and Create an EKS Cluster from the CLI (see page 1271) for more details.

6.12.7.6.4.2 Export Variables to Use as Flags in Cluster Creation

Below is an AWS ECR example:

```
export REGISTRY_URL=<ecr-registry-URI>
```

- `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- NOTE: Other local registries may use the options below:
  - JFrog - `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
  - `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
  - `REGISTRY_PASSWORD` : optional if username is not set.

For more information to set up a private registry with a registry mirror, see this page (see page 1609) for details on using that flag.

Next Topic

AWS Registry Configuration (see page 1180)

6.12.7.7 AWS Registry Configuration

Configure your cluster to use an existing local registry (see page 1609) when attempting to pull images by adding the flag(s) to the `dkp create cluster` command to pull images from your local registry.

If you set the `--registry-mirror` flag during cluster creation, the Kubelet will now send to requests to the dynamic-credential-provider with a different config. Only use one image registry per cluster.

To apply private registry configurations during the `dkp cluster create` operation, add the appropriate flags to the command:

Registry configuration	Flag
CA certificate chain to use while communicating with the registry mirror using Transport Layer Security(TLS)	`--registry-mirror-cacert file`

Registry configuration	Flag
URL of a container registry to use as a mirror in the cluster	`--registry-mirror-url string` OR apply variable `${REGISTRY_URL}`
Set to a user that has pull access to this registry	`--registry-mirror-username string` OR apply variable `${REGISTRY_USERNAME}`
Password to authenticate the registry mirror	`--registry-mirror-password string` OR apply variable `${REGISTRY_PASSWORD}`

This is useful when using an internal registry and when Internet access is not available such as in an air-gapped environment. However, registry mirrors can be used in non-air-gapped environments as well for security and speed.

AWS ECR - Adding the mirror flags to EKS would enable new clusters to also use ECR as image mirror. If you set the `--registry-mirror` flag, the Kubelet will now send to requests to the `dynamic-credential-provider` with a different config. You can still pull your own images from ECR directly or use ECR as a mirror.

When the cluster is up and running, you can deploy and test workloads.

6.12.7.7.1 Registry Mirror Cluster Example

Selecting your provider, run:

```
dkp create cluster [aws, azure, gcp, preprovisoned, vsphere] \
    --cluster-name=${CLUSTER_NAME} \
    --registry-mirror-cacert /tmp/registry.pem \
    --registry-mirror-url=${REGISTRY_URL}
```

More information is found in the Custom Installation and Additional Infrastructure Tools (see page 1050) sections under the Create a New Cluster section of each Infrastructure Provider. Mirrors can be used in both air-gapped and non-air-gapped environments by adding the flag to the `dkp create cluster` command.

6.12.7.7.2 Next Topic

AWS HTTP Proxy (see page 1182)

If none of the customizations apply, continue to installation instructions for your environment:

- AWS Install in a Non-air-gapped Environment (see page 1184)
  OR
- AWS Install in an Air-gapped Environment (see page 1207)

6.12.7.8 AWS HTTP Proxy

When creating a DKP cluster in environments that use an HTTP/HTTPS proxy, you must provide proxy details. The proxy values are strings that list a set of proxy servers, URLs, or wildcard addresses that is specific to your environment.

To create a proxied environment, you need to include flags at various action item points:

- Bootstrap cluster
- CAPI components
- Cluster creation
- DKP Kommander component

Create the bootstrap cluster and CAPI components using the appropriate commands, `dkp create bootstrap` and `dkp create capi-components` respectively, combined with the command line flags to include your HTTP/S proxy information.

You can also specify HTTP/S proxy information in an override file when using Konvoy Image Builder (KIB) (see page 1626).

Without these values provided as part of the relevant `dkp create` command, DKP cannot create the requisite parts of your new cluster correctly. This is true of both management and managed clusters (see page 79) alike.

For full HTTP Proxy configuration, you need to specify proxy settings using all the details in the Configuring an HTTP/HTTPS Proxy (see page 1053) section of the documentation for:

- Configure the Bootstrap Cluster HTTP Proxy Settings (see page 1054)
- Create CAPI Components with HTTP/HTTPS Proxy Settings (see page 1055)
- Create a Cluster with HTTP/HTTPS Proxy (see page 1056)
- Configure an HTTP/HTTPS Proxy for the DKP Kommander Component (see page 1058)

AWS Example:

```
dkp create cluster aws \
  --cluster-name ${CLUSTER_NAME} \
  --control-plane-http-proxy="${CONTROL_PLANE_HTTP_PROXY}" \
  --control-plane-https-proxy="${CONTROL_PLANE_HTTPS_PROXY}" \
  --control-plane-no-proxy="${CONTROL_PLANE_NO_PROXY}" \
  --worker-http-proxy="${WORKER_HTTP_PROXY}" \
  --worker-https-proxy="${WORKER_HTTPS_PROXY}" \
  --worker-no-proxy="${WORKER_NO_PROXY}"
```

6.12.7.8.1 Next Topic

AWS Output Directory YAML (see page 1183)

If none of the customizations apply, continue to installation instructions for your environment:

- AWS Install in a Non-air-gapped Environment (see page 1184)
  OR
- AWS Install in an Air-gapped Environment (see page 1207)

6.12.7.9 AWS Output Directory YAML

You can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag used with `--output=json|yaml`. You create the directory of where to output resources to files.

Using this flag will create multiple files in the specified directory which must already exist:

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--ami=${AWS_AMI_ID} \
--dry-run \
--output=yaml \
--output-directory=<existing-directory>
```

For more information regarding this flag or others, please refer to the CLI section of the documentation for the dkp create cluster (see page 1853) command and select your provider.

6.12.7.9.1 Next Topic

AWS Provision on the Flatcar Linux OS (see page 1183)

If none of the customizations apply, continue to installation instructions for your environment:

- AWS Install in a Non-air-gapped Environment (see page 1184)
  OR
- AWS Install in an Air-gapped Environment (see page 1207)

6.12.7.10 AWS Provision on the Flatcar Linux OS

Flatcar default network interface name might require specifying. It is most likely to be `ens192` requiring passing the parameter `--virtual-ip-interface ens192` to the `dkp create cluster aws`

command. Otherwise, the cluster creation might fail because `kube-vip` can not configure the first control-plane virtual IP.

6.12.7.10.1 Flatcar Linux example

These flags are also shown in context on the Create Cluster page for either air-gapped (see page 1218) or non-air-gapped (see page 1190) environments:

```
dkp create cluster aws \
    --cluster-name ${CLUSTER_NAME} \
    --os-hint flatcar
```

For provisioning DKP on Flatcar, DKP configures cluster nodes to use Control Groups (cgroups) version 1[849]. In versions prior to Flatcar 3033.3.x, a restart is required in order to apply the changes to the kernel. For more information, refer to the Flatcar documentation[850].

Also note that once Ignition[851] runs, it is not available on reboot.

6.12.7.10.2 Next Topic

Continue to installation instructions for your environment:

- AWS Install in a Non-air-gapped Environment (see page 1184)
  OR
- AWS Install in an Air-gapped Environment (see page 1207)

6.12.8 AWS Install in a Non-air-gapped Environment

This section provides instructions to install DKP in an AWS non-air-gapped environment with custom settings. First you create an Bootstrap Cluster (see page 1187) and then move the CAPI resources to the workload cluster and delete the bootstrap cluster.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

849 https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v1/cgroups.html#what-are-cgroups
850 https://www.flatcar.org/docs/latest/container-runtimes/switching-to-unified-cgroups/#starting-new-nodes-with-legacy-cgroups
851 https://www.flatcar.org/docs/latest/provisioning/ignition/#ignition-only-runs-once

6.12.8.1 AWS Specific Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create permissions and roles on the Minimal Permissions and Role to Create Clusters (see page 1156) page.
2. Create Cluster IAM Policies and Roles (see page 1162).
3. Export the AWS region where you want to deploy the cluster:

   ```
   export AWS_REGION=us-west-2
   ```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

   ```
   export AWS_PROFILE=<profile>
   ```

If using AWS ECR as your local private registry, more information can be found on the Registry Mirror Tools (see page 1606) page.

To deploy a cluster with a custom image in a region where CAPI images[852] are not provided, you need to use Konvoy Image Builder (see page 1153) to create your own image for the region.

For multi-tenancy, every tenant should be in a different AWS account to ensure they are truly independent of other tenants in order to enforce security.

6.12.8.1.1 Next Step

Custom AMI in Cluster Creation (see page 1185)

6.12.8.2 Custom AMI in Cluster Creation

The default AWS image is not recommended for use in production. D2iQ suggests using Konvoy Image Builder (see page 1629) to create a custom AMI and take advantage of enhanced cluster operations.

852 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html

- ⚠️ **IMPORTANT:** The AMI must be created with Konvoy Image Builder (see page 1626) in order to use the registry mirror feature.

Depending on which version of DKP (see page 1626) you are running, steps and flags will be different. To deploy in a region where CAPI images are not provided, you need to use KIB to create your own image for the region. For a list of supported AWS regions, refer to the Published AMI[853]information from AWS.

6.12.8.2.1 Begin image creation:

Run the `konvoy-image` command to build and validate the image.

```
konvoy-image build aws images/ami/rhel-86.yaml
```

By default, it builds in the `us-west-2` region. to specify another region set the `--region` flag as shown in the command below:

```
konvoy-image build aws --region us-east-1 images/ami/rhel-86.yaml
```

Ensure you have named the correct AMI image YAML file for your OS[854] in the `konvoy-image build` command.

After KIB provisions the image successfully, the `ami` id is printed and written to the `packer.pkr.hcl` (Packer config) file. This file has an `artifact_id` field whose value provides the name of the AMI ID as shown in the example below. That is the `ami` you use in the `dkp create cluster` command:


```
{
  "builds": [
    {
      "name": "kib_image",
      "builder_type": "amazon-ebs",
      "build_time": 1698086886,
      "files": null,
      "artifact_id": "us-west-2:ami-04b8dfef8bd33a016",
      "packer_run_uuid": "80f8296c-e975-d394-45f9-49ef2ccc6e05",
      "custom_data": {
        "containerd_version": "",
        "distribution": "RHEL",
        "distribution_version": "8.6",
```


853 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html
854 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ami

```
        "kubernetes_cni_version": "",
        "kubernetes_version": "1.26.6"
      }
    }
  ],
  "last_run_uuid": "80f8296c-e975-d394-45f9-49ef2ccc6e05"
}
```

To use a custom AMI when creating your cluster (see page 1190), you must create that AMI using KIB first. Then perform the export and name the custom AMI for use in the command `dkp create cluster`:

```
export AWS_AMI_ID=ami-<ami-id-here>
```

Inside the sections for either Non-air-gapped (see page 1190) or Air-gapped (see page 1218) cluster creation, you will find the instructions for how to apply custom images.

The export command is similar to this:

```
export AWS_AMI_ID=ami-ami-04b8dfef8bd33a016
```

The create cluster command would then be:

```
dkp create cluster aws --ami=${AWS_AMI_ID}
```

More information about using the custom AMI in cluster creation can be found here: Use Custom AMI to Build Cluster (see page 1640).

6.12.8.2.2 Next Step

AWS Bootstrap Cluster (see page 1187)

6.12.8.3 AWS Bootstrap Cluster

To create Kubernetes clusters, Konvoy uses Cluster API[855] (CAPI) controllers. These controllers run on a Kubernetes cluster. To get started, you need a *bootstrap* cluster. By default, Konvoy creates a bootstrap cluster for you in a Docker container using the Kubernetes-in-Docker (KIND[856]) tool.

6.12.8.3.1 Prerequisites

Before you begin, you must:

- Complete the steps in Prerequisites (see page 1151).
- Ensure the `dkp` binary can be found in your $PATH.

855 https://cluster-api.sigs.k8s.io/
856 https://github.com/kubernetes-sigs/kind

6.12.8.3.1.1 Bootstrap Cluster Lifecycle Services

1. Review Universal Configurations for all Infrastructure Providers (see page 1052) regarding settings, flags and other choices and then begin bootstrapping.
2. Create a bootstrap cluster:

```
dkp create bootstrap --kubeconfig $HOME/.kube/config
```

   - Configuring an HTTP/HTTPS Proxy (see page 1053) use `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful.

HTTP Flags if needed:

To create a bootstrap cluster in a proxied environment (see page 1054) use the flags for HTTP:

```
--http-proxy <string> \
--https-proxy <string> \
--no-proxy <string>
```

Using a Custom AWS CA

You need to add the custom CAs into two places:

- The `capa-controller-manager` pod, because CAPA controllers interact with AWS API when creating and deleting infrastructure.
- The trusted root CAs in the AWS AMI used as Kubernetes nodes. The first step of the node bootstrap process is to fetch the sensitive information from the AWS Secrets Manager service, so the `aws` client on the instances needs to trust this custom CA. This process is unique to your environment but a general flow can be similar to what is documented in adding trusted root certificates to the server[857].

1. Place the AWS CA file as `ca.pem` in your working directory
2. Create a ConfigMap with the contents of the file:

```
kubectl create configmap -n capa-system aws-ca --from-file=ca.pem
```

3. Update the capa-controller-manager to set an environment variable `AWS_CA_BUNDLE` in `capa-controller-manager`:

857 https://manuals.gfi.com/en/kerio/connect/content/server-configuration/ssl-certificates/adding-trusted-root-certificates-to-the-server-1605.html

```
kubectl patch deployment -n capa-system capa-controller-manager --patch
'{"spec":{"template":{"spec":{"$setElementOrder/containers":
[{"name":"manager"},{"name":"kube-rbac-proxy"}],"$setElementOrder/volumes":
[{"name":"cert"},{"name":"credentials"},{"name":"aws-ca"}],"containers":
[{"$setElementOrder/env":[{"name":"AWS_SHARED_CREDENTIALS_FILE"},
{"name":"AWS_CA_BUNDLE"}],"$setElementOrder/volumeMounts":[{"mountPath":"/tmp/
k8s-webhook-server/serving-certs"},{"mountPath":"/home/.aws"},{"mountPath":"/
home/.konvoy/aws-ca.pem"}],"env":[{"name":"AWS_CA_BUNDLE","value":"/
home/.konvoy/aws-ca.pem"}],"name":"manager","volumeMounts":[{"mountPath":"/
home/.konvoy/aws-ca.pem","name":"aws-ca","subPath":"ca.pem"}]}],"volumes":
[{"configMap":{"name":"aws-ca"},"name":"aws-ca"}]}}}}'
```

Output:

```
✓ Creating a bootstrap cluster
✓ Initializing new CAPI components
```

Konvoy creates a bootstrap cluster using KIND[858] as a library and deploys Cluster API[859] providers on the cluster. Refer to Customizing CAPI Components for a Cluster (see page 1061)for more details.

Konvoy waits until the controller-manager and webhook deployments of these providers are ready.

List these deployments using this command:

```
kubectl get --all-namespaces deployments -l=clusterctl.cluster.x-k8s.io
```

Output:

```
NAMESPACE                           NAME
READY   UP-TO-DATE   AVAILABLE   AGE
capa-system                         capa-controller-manager                          1
/1     1            1           2m8s
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-manager        1
/1     1            1           2m10s
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-manager    1
/1     1            1           2m10s
capi-system                         capi-controller-manager                          1
/1     1            1           2m11s
cappp-system                        cappp-controller-manager                         1
/1     1            1           2m6s
capv-system                         capv-controller-manager                          1
/1     1            1           2m5s
capz-system                         capz-controller-manager                          1
/1     1            1           2m7s
```

858 https://github.com/kubernetes-sigs/kind
859 https://cluster-api.sigs.k8s.io/

```
cert-manager                        cert-manager                                1
/1     1            1           2m21s
cert-manager                        cert-manager-cainjector                     1
/1     1            1           2m21s
cert-manager                        cert-manager-webhook                        1
/1     1            1           2m21s
```

6.12.8.3.1.2 Next Step

AWS Create a New Customized Cluster (see page 1190)

6.12.8.4 AWS Create a New Customized Cluster

6.12.8.4.1 Prerequisites

Before you begin, make sure you have created a Bootstrap (see page 1187) cluster.

> By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

6.12.8.4.1.1 Name Your Cluster

The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[860] for more naming information. Give your cluster a unique name suitable for your environment.

In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name. You will set this environment variable during cluster creation below.

⚠ **IMPORTANT:** Ensure your subnets (see page 1065) do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the

kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
```

860 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
    cidrBlocks:
    - 10.96.0.0/12
```

For multi-tenancy, every tenant should be in a different AWS account to ensure they are truly independent of other tenants in order to enforce security.

6.12.8.4.2 Create a New AWS Kubernetes Cluster

1. Ensure your AWS credentials are up to date. If you are using Static Credentials, use the following command to refresh the credentials. Otherwise, proceed to step 2:

   ```
   dkp update bootstrap credentials aws
   ```

2. Set the **environment variables** of *cluster name* selected according to requirements and *custom AMI* identification:

   ```
   export CLUSTER_NAME=<aws-example>
   export AWS_AMI_ID=<ami-...>
   ```

In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1626).

There are two approaches to supplying the ID of your AMI. Either provide the ID of the AMI or provide a way for DKP to discover the AMI using location, format and OS information:

- **Option One** - Provide the ID of your AMI:
  - Use the example command below leaving the existing flag that provides the AMI ID: `--ami AMI_ID`
- **Option Two** - Provide a path for your AMI with the information required for image discover:
  a. Where the AMI is published using your AWS Account ID: `--ami-owner AWS_ACCOUNT_ID`
  b. The format or string used to search for matching AMIs and ensure it references the Kubernetes version plus the base OS name: `--ami-base-os ubuntu-20.04`
  c. The base OS information: `--ami-format 'example-{{.BaseOS}}-?{{.K8sVersion}}-*'`

- **(Optional)** Use a registry mirror (see page 1609). Configure your cluster to use an existing local registry (see page 1606) as a mirror when attempting to pull images previously pushed to your registry (see page 1175).
  Set an environment variable with your registry address for ECR:

  ```
  export REGISTRY_URL=<ecr-registry-URI>
  ```

MORE REGISTRY EXPORT COMMANDS for later use with flags during cluster creation:

For other registries, more environment variables are:

```
export REGISTRY_URL=<registry-address>:<registry-port>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
export REGISTRY_CA=<path to the cacert file on the bastion>
```

Definitions:

- `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.

Other local registries may use the options below:

- JFrog - `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional-set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.
- To increase Docker Hub's rate limit[861] use your Docker Hub credentials when creating the cluster, by setting flags `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username=<your-username> --registry-mirror-password=<your-password>` when running `dkp create cluster` .

EKS with AWS ECR - Adding the mirror flags to EKS would enable new clusters to also use ECR as image mirror. If you set the `--registry-mirror` flag, the Kubelet will now send to requests to the `dynamic-credential-provider` with a different config. You can still pull your own images from ECR directly or use ECR as a mirror.

4. Generate the Kubernetes cluster objects with a `dry run` . The following example shows a common configuration. See dkp create cluster aws (see page 1856) reference for the full list of cluster creation options:

   ```
   dkp create cluster aws \
   ```

861 https://docs.docker.com/docker-hub/download-rate-limit/

```
--cluster-name=${CLUSTER_NAME} \
--ami=${AWS_AMI_ID} \
--dry-run \
--output=yaml \
> ${CLUSTER_NAME}.yaml
```

Expand the drop-downs for more flags for use in cluster creation such as **registry, HTTP, FIPS and other flags** to apply in the step above.

Flatcar

Flatcar OS (see page 1183) use this flag used to instruct the bootstrap cluster to make some changes related to the installation paths:

```
--os-hint flatcar
```

If using a REGISTRY MIRROR, use these FLAGS in your create cluster command:

```
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD}
```

See also: Use a Registry Mirror (see page 1609)

Applying exported image variable

Don't forget, as explained in theCustom AMI in Cluster Creation (see page 1185) before you bootstrapped, you need to apply the flag for your image in cluster creation.

```
--ami=${AWS_AMI_ID}
```

FIPS Requirements

To create a cluster in FIPS mode (see page 1598), inform the controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes by adding those flags to `dkp create cluster` command:

```
--kubernetes-version=v1.28.7+fips.0 \
--etcd-version=3.5.10+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
```

HTTP ONLY

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

```
--http-proxy <<http proxy list>>
--https-proxy <<https proxy list>>
--no-proxy <<no proxy list>>
```

- To configure the Control Plane and Worker nodes to use an HTTP proxy:

```
export CONTROL_PLANE_HTTP_PROXY=http://example.org:8080
export CONTROL_PLANE_HTTPS_PROXY=http://example.org:8080
export
CONTROL_PLANE_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1
,10.96.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.s
vc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.sv
c.cluster,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"

export WORKER_HTTP_PROXY=http://example.org:8080
export WORKER_HTTPS_PROXY=http://example.org:8080
export
WORKER_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96.
0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kube
rnetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.clust
er,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"
```

- Replace:
  - `example.org,example.com,example.net` with you internal addresses
  - `localhost` and `127.0.0.1` addresses should not use the proxy
  - `10.96.0.0/12` is the default Kubernetes service subnet
  - `192.168.0.0/16` is the default Kubernetes pod subnet
  - `kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local` is the internal Kubernetes kube-apiserver service
  - `.svc,.svc.cluster,.svc.cluster.local` is the internal Kubernetes services
  - `169.254.169.254` is the AWS metadata server
  - `.elb.amazonaws.com` is for the worker nodes to allow them to communicate directly to the kube-apiserver ELB
- Use the HTTP flags in the `dkp create cluster` command:

```
--control-plane-http-proxy=${CONTROL_PLANE_HTTP_PROXY} \
```

```
--control-plane-https-proxy=${CONTROL_PLANE_HTTPS_PROXY} \
--control-plane-no-proxy=${CONTROL_PLANE_NO_PROXY} \
--worker-http-proxy=${WORKER_HTTP_PROXY} \
--worker-https-proxy=${WORKER_HTTPS_PROXY} \
--worker-no-proxy=${WORKER_NO_PROXY}
```

Individual manifests using the Output Directory flag:

You can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag. This will create multiple files in the specified directory which must already exist:

```
--output-directory=<existing-directory>
```

Refer to the Cluster Creation Customization Choices (see page 1170) section for more information on how to use optional flags such as the `--output-directory` flag.

5. Inspect or edit the cluster objects and familiarize yourself with Cluster API before editing the cluster objects as edits can prevent the cluster from deploying successfully. See AWS Customizing CAPI Clusters (see page 1173).
6. **(Optional)** Modify Control Plane Audit logs - Users can make modifications to the `KubeadmControlplane` cluster-api object to configure different `kubelet` options. See the following guide (see page 1613) if you wish to configure your control plane beyond the existing options that are available from flags.
7. Create the cluster from the objects generated with the `dry run`. A warning will appear in the console if the resource already exists and will require you to remove the resource or update your YAML.

   ```
   kubectl create -f ${CLUSTER_NAME}.yaml
   ```

   NOTE: If you used the `--output-directory` flag in your `dkp create .. --dry-run` step above, create the cluster from the objects you created by specifying the directory:

   ```
   kubectl create -f <existing-directory>/
   ```

OUTPUT will be similar to output shown in this drop-down:

```
cluster.cluster.x-k8s.io/aws-example created
awscluster.infrastructure.cluster.x-k8s.io/aws-example created
kubeadmcontrolplane.controlplane.cluster.x-k8s.io/aws-example-control-plane created
awsmachinetemplate.infrastructure.cluster.x-k8s.io/aws-example-control-plane created
secret/aws-example-etcd-encryption-config created
machinedeployment.cluster.x-k8s.io/aws-example-md-0 created
awsmachinetemplate.infrastructure.cluster.x-k8s.io/aws-example-md-0 created
```

```
kubeadmconfigtemplate.bootstrap.cluster.x-k8s.io/aws-example-md-0 created
clusterresourceset.addons.cluster.x-k8s.io/calico-cni-installation-aws-example
created
configmap/calico-cni-installation-aws-example created
configmap/tigera-operator-aws-example created
clusterresourceset.addons.cluster.x-k8s.io/aws-ebs-csi-aws-example created
configmap/aws-ebs-csi-aws-example created
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-aws-example created
configmap/cluster-autoscaler-aws-example created
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-aws-example created
configmap/node-feature-discovery-aws-example created
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-aws-example
created
configmap/nvidia-feature-discovery-aws-example created
```

8. Wait for the cluster control-plane to be ready:

   ```
   kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
   timeout=20m
   ```

   Output:

   ```
   cluster.cluster.x-k8s.io/aws-example condition met
   ```

   The `READY` status becomes `True` after the cluster control-plane becomes ready in one of the following steps.

9. After the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. Konvoy provides a command to describe the current status of the cluster:

   ```
   dkp describe cluster -c ${CLUSTER_NAME}
   ```

OUTPUT:

```
NAME                                                          READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/aws-example                                           True
60s
├─ClusterInfrastructure - AWSCluster/aws-example              True
5m23s
├─ControlPlane - KubeadmControlPlane/aws-example-control-plane  True
60s
│ ├─Machine/aws-example-control-plane-55jh4                   True
4m59s
```

```
| ├─Machine/aws-example-control-plane-6sn97                    True
2m49s
| └─Machine/aws-example-control-plane-nx9v5                    True
66s
└─Workers
  └─MachineDeployment/aws-example-md-0                         True
117s
    ├─Machine/aws-example-md-0-cb9c9bbf7-hcl8z                 True
3m1s
    ├─Machine/aws-example-md-0-cb9c9bbf7-rtdqw                 True
3m2s
    ├─Machine/aws-example-md-0-cb9c9bbf7-t894m                 True
3m1s
    └─Machine/aws-example-md-0-cb9c9bbf7-td29r                 True
```

10. As they progress, the controllers also create Events. List the Events using this command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, the example uses `grep`. It is also possible to use separate commands to get Events for specific objects. For example, `kubectl get events --field-selector involvedObject.kind="AWSCluster"` and `kubectl get events --field-selector involvedObject.kind="AWSMachine"`.

OUTPUT will be similar to output shown in this drop-down:

```
7m26s       Normal    SuccessfulSetNodeRef                            machine/aws-
example-control-plane-2wb9q      ip-10-0-182-218.us-west-2.compute.internal
11m         Normal    SuccessfulCreate                                awsmachine/aws-
example-control-plane-vcjkr   Created new control-plane instance with id
"i-0dde024e80ae3de7a"
11m         Normal    SuccessfulAttachControlPlaneELB                 awsmachine/aws-
example-control-plane-vcjkr   Control plane instance "i-0dde024e80ae3de7a" is
registered with load balancer
7m25s       Normal    SuccessfulDeleteEncryptedBootstrapDataSecrets   awsmachine/aws-
example-control-plane-vcjkr   AWS Secret entries containing userdata deleted
7m6s        Normal    FailedDescribeInstances                         awsmachinepool/
aws-example-mp-0              No Auto Scaling Groups with aws-example-mp-0 found
7m3s        Warning   FailedLaunchTemplateReconcile                   awsmachinepool/
aws-example-mp-0              Failed to reconcile launch template: ValidationError:
AutoScalingGroup name not found - AutoScalingGroup aws-example-mp-0 not found
74s         Warning   FailedLaunchTemplateReconcile                   awsmachinepool/
aws-example-mp-0              (combined from similar events): Failed to reconcile
launch template: ValidationError: AutoScalingGroup name not found - AutoScalingGroup
aws-example-mp-0 not found
16m         Normal    SuccessfulCreateVPC                             awscluster/aws-
example                       Created new managed VPC "vpc-032fff0fe06a85035"
```

```
16m         Normal    SuccessfulSetVPCAttributes                    awscluster/aws-
example                       Set managed VPC attributes for "vpc-032fff0fe06a85035"
16m         Normal    SuccessfulCreateSubnet                        awscluster/aws-
example                       Created new managed Subnet "subnet-0677a4fbd7d170dfe"
16m         Normal    SuccessfulModifySubnetAttributes              awscluster/aws-
example                       Modified managed Subnet "subnet-0677a4fbd7d170dfe"
 attributes
16m         Normal    SuccessfulCreateSubnet                        awscluster/aws-
example                       Created new managed Subnet "subnet-04fc9deb4fa9f8333"
16m         Normal    SuccessfulCreateInternetGateway               awscluster/aws-
example                       Created new managed Internet Gateway
"igw-07cd7ad3e6c7c1ca7"
16m         Normal    SuccessfulAttachInternetGateway               awscluster/aws-
example                       Internet Gateway "igw-07cd7ad3e6c7c1ca7" attached to
VPC "vpc-032fff0fe06a85035"
16m         Normal    SuccessfulCreateNATGateway                    awscluster/aws-
example                       Created new NAT Gateway "nat-0a0cf17d29150cf9a"
13m         Normal    SuccessfulCreateRouteTable                    awscluster/aws-
example                       Created managed RouteTable "rtb-09f4e2eecb7462d22"
13m         Normal    SuccessfulCreateRoute                         awscluster/aws-
example                       Created route {
13m         Normal    SuccessfulAssociateRouteTable                 awscluster/aws-
example                       Associated managed RouteTable "rtb-09f4e2eecb7462d22"
 with subnet "subnet-0677a4fbd7d170dfe"
13m         Normal    SuccessfulCreateRouteTable                    awscluster/aws-
example                       Created managed RouteTable "rtb-0007b98b36f37d1e4"
13m         Normal    SuccessfulCreateRoute                         awscluster/aws-
example                       Created route {
13m         Normal    SuccessfulAssociateRouteTable                 awscluster/aws-
example                       Associated managed RouteTable "rtb-0007b98b36f37d1e4"
 with subnet "subnet-04fc9deb4fa9f8333"
13m         Normal    SuccessfulCreateRouteTable                    awscluster/aws-
example                       Created managed RouteTable "rtb-079a1d7d3667c2525"
13m         Normal    SuccessfulCreateRoute                         awscluster/aws-
example                       Created route {
13m         Normal    SuccessfulAssociateRouteTable                 awscluster/aws-
example                       Associated managed RouteTable "rtb-079a1d7d3667c2525"
 with subnet "subnet-0a266c15dd211ce6c"
13m         Normal    SuccessfulCreateRouteTable                    awscluster/aws-
example                       Created managed RouteTable "rtb-0e5ebc8ec29848a17"
13m         Normal    SuccessfulCreateRoute                         awscluster/aws-
example                       Created route {
13m         Normal    SuccessfulAssociateRouteTable                 awscluster/aws-
example                       Associated managed RouteTable "rtb-05a05080bbb3cead9"
 with subnet "subnet-0725068cca16ad9f9"
13m         Normal    SuccessfulCreateSecurityGroup                 awscluster/aws-
example                       Created managed SecurityGroup "sg-0379bf77211472854"
for Role "bastion"
13m         Normal    SuccessfulCreateSecurityGroup                 awscluster/aws-
example                       Created managed SecurityGroup "sg-0a4e0635f68a2f57d"
for Role "apiserver-lb"
```

```
13m         Normal    SuccessfulAuthorizeSecurityGroupIngressRules    awscluster/aws-
example                       Authorized security group ingress rules [protocol=tcp/
range=[5473-5473]/description=typha (calico) protocol=tcp/range=[179-179]/
description=bgp (calico) protocol=4/range=[-1-65535]/description=IP-in-IP (calico)
protocol=tcp/range=[22-22]/description=SSH protocol=tcp/range=[6443-6443]/
description=Kubernetes API protocol=tcp/range=[2379-2379]/description=etcd
protocol=tcp/range=[2380-2380]/description=etcd peer] for SecurityGroup
"sg-00db2e847c0b49d6e"
13m         Normal    SuccessfulAuthorizeSecurityGroupIngressRules    awscluster/aws-
example                       Authorized security group ingress rules [protocol=tcp/
range=[5473-5473]/description=typha (calico) protocol=tcp/range=[179-179]/
description=bgp (calico) protocol=4/range=[-1-65535]/description=IP-in-IP (calico)
protocol=tcp/range=[22-22]/description=SSH protocol=tcp/range=[30000-32767]/
description=Node Port Services protocol=tcp/range=[10250-10250]/description=Kubelet
API] for SecurityGroup "sg-01fe3426404f94708"
```

DKP uses AWS CSI as the default storage provider (see page 110). You can use a Kubernetes CSI[862] compatible storage solution that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[863] for more information.

If you're not using the default, you cannot deploy an alternate provider until ***after*** the `dkp create cluster` is finished. However, it must be determined before Kommander installation.

6.12.8.4.2.1 Known Limitations

Be aware of these limitations in the current release of Konvoy.

- The Konvoy version used to create a bootstrap cluster must match the Konvoy version used to create a workload cluster.
- Konvoy supports deploying one workload cluster.
- Konvoy generates a set of objects for one Node Pool.
- Konvoy does not validate edits to cluster objects.

6.12.8.4.2.2 Related Topics

Creating DKP Non-air-gapped Clusters from the UI (see page 1200)

862 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
863 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

6.12.8.4.2.3 Next Step

Make the AWS Non-air-gapped Cluster Self-Managed (see page 1201)

6.12.8.4.3 Creating DKP Non-air-gapped Clusters from the UI

A guide for creating DKP clusters on AWS console

6.12.8.4.3.1 Prerequisites

Configure an AWS Infrastructure (see page 1148), or add credentials using AWS role credentials (see page 625).

Simplified Cluster Creation on AWS in the UI

1. In the selected workspace Dashboard, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
2. On the Add Cluster page, select the **Create DKP Cluster**.
3. Provide some basic cluster details within the form:
    - **Workspace**: The workspace where this cluster belongs (if within the Global workspace).
    - **Kubernetes Version**: The initial version of Kubernetes to install on the cluster.
    - **Name**: A valid Kubernetes name for the cluster.
    - **Add Labels**: By default, your cluster has labels that reflect the infrastructure provider provisioning. For example, your AWS cluster may have a label for the data center region and `provider: aws`. Cluster labels are matched to the selectors created for Projects (see page 716). Changing a cluster label may add or remove the cluster from projects.
4. Select the pre-configured AWS Infrastructure (see page 1148), or AWS role credentials (see page 625) to display the remaining options specific to AWS.
    - **Region**: Select a data center region or specify a custom region.
    - **Configure Node Pools**: Specify pools of nodes, their machine types, quantity, and the IAM instance profile.
        - **Machine Type**: Machine instance type.
        - **Quantity**: Number of nodes. The control plane must be an odd number.
        - **IAM instance profile**: Name of the IAM instance profile to assign to the machines.
    - **AMI Image**: You specify the AMI as part of each pool by using either group of fields, but not both groups:
        - (a) AMI ID specifying by name.
            - **AMI ID**: AMI ID to use for all nodes.

        - (b) AMI ID for lookup - Owner ID, Base OS, Lookup Format (all fields are required, if any is used)
            - **Base OS**: Base OS for Lookup search.
            - **Lookup Format**: Lookup Format string to generate AMI search name from.
            - **Owner ID**: Owner ID for AMI Lookup search.
    - **Worker Availability Zone:** Worker Zones for a region (worker nodes only).
    - **Add Infrastructure Provider Tags**: Specify tags applied on all resources created in your infrastructure for this cluster. Different infrastructure providers have varying restrictions on the usable tags. See the AWS Tags User Guide[864] for more information on using tags in AWS.
5. Select **Create** to begin provisioning the cluster. This step may take a few minutes, taking time for the cluster to be ready and fully deploy its components. The cluster automatically tries to join, and should resolve after it is fully-provisioned.

Next Step

Make the AWS Non-air-gapped Cluster Self-Managed (see page 1201)

6.12.8.5 Make the AWS Non-air-gapped Cluster Self-Managed

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which then deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, which makes the workload cluster self-managed. This section describes how to make a workload cluster self-managed.

This page contains instructions on how to make your cluster self-managed. This is necessary if there is only one cluster in your environment, or if this cluster should become the Management cluster in a multi-cluster environment.

If you already have a self-managed or Management cluster in your environment, skip this page.

6.12.8.5.1 Make the New Kubernetes Cluster Manage Itself

1. Deploy cluster lifecycle services on the workload cluster:

    ```
    dkp create capi-components --kubeconfig ${CLUSTER_NAME}.conf
    ```

    If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful.

864 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Using_Tags.html

More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

```
✓ Initializing new CAPI components
```

2. Move the Cluster API objects from the bootstrap to the workload cluster:

   The cluster lifecycle services on the workload cluster are ready, but the workload cluster configuration is on the bootstrap cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the bootstrap to the workload cluster. This process is also called a Pivot[865].

   ```
   dkp move capi-resources --to-kubeconfig ${CLUSTER_NAME}.conf
   ```

   ```
    ✓ Moving cluster resources
   You can now view resources in the moved cluster by using the --kubeconfig flag
   with kubectl. For example: kubectl --kubeconfig=aws-example.conf get nodes
   ```

   NOTE: To ensure only one set of cluster lifecycle services manages the workload cluster, Konvoy first pauses reconciliation of the objects on the bootstrap cluster, then creates the objects on the workload cluster. As Konvoy copies the objects, the cluster lifecycle services on the workload cluster reconcile the objects. The workload cluster becomes self-managed after Konvoy creates all the objects. If it fails, the `move` command can be safely retried.

3. Wait for the cluster control-plane to be ready:

   ```
   kubectl --kubeconfig ${CLUSTER_NAME}.conf wait --for=condition=ControlPlaneRead
   y "clusters/${CLUSTER_NAME}" --timeout=20m
   ```

   ```
   cluster.cluster.x-k8s.io/aws-example condition met
   ```

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

865 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use Konvoy with the workload cluster kubeconfig.

```
dkp describe cluster --kubeconfig ${CLUSTER_NAME}.conf -c ${CLUSTER_NAME}
```

```
NAME                                                              READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/aws-example                                               True
109s
├─ClusterInfrastructure - AWSCluster/aws-example                  True
112s
├─ControlPlane - KubeadmControlPlane/aws-example-control-plane    True
109s
│ ├─Machine/aws-example-control-plane-55jh4                       True
111s
│ ├─Machine/aws-example-control-plane-6sn97                       True
111s
│ └─Machine/aws-example-control-plane-nx9v5                       True
110s
└─Workers
  └─MachineDeployment/aws-example-md-0                            True
114s
    ├─Machine/aws-example-md-0-cb9c9bbf7-hcl8z                    True
111s
    ├─Machine/aws-example-md-0-cb9c9bbf7-rtdqw                    True
111s
    ├─Machine/aws-example-md-0-cb9c9bbf7-t894m                    True
111s
    └─Machine/aws-example-md-0-cb9c9bbf7-td29r                    True
111s
```

5. Remove the bootstrap cluster, as the workload cluster is now self-managed:

```
dkp delete bootstrap
```

```
 ✓ Deleting bootstrap cluster
```

6.12.8.5.2 Known Limitations

- Before making a workload cluster self-managed, be sure that its control plane nodes have sufficient permissions for running Cluster API controllers. See IAM Policy Configuration (see page 1162).
- Konvoy supports moving only one set of cluster objects from the bootstrap cluster to the workload cluster, or vice-versa.
- Konvoy only supports moving all namespaces in the cluster; Konvoy does not support migration of individual namespaces.

6.12.8.5.3 Next Step

Explore the New AWS Non-air-gapped Cluster (see page 1204)

6.12.8.6 Explore the New AWS Non-air-gapped Cluster

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster.

Follow these steps:

1. Fetch the kubeconfig file with the command:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

2. Verify the API server is up by listing the nodes:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
```

Wait for the Status to move to `Ready` while `calico-node` pods are being deployed.

3. List the Pods with the command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get pods -A
```

6.12.8.6.1 Next Step

AWS Install Kommander Non-air-gapped (see page 1205)

6.12.8.7 AWS Install Kommander Non-air-gapped

This section provides installation instructions for the Kommander component of DKP for a non-air-gapped environment in AWS.

6.12.8.7.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

6.12.8.7.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init > kommander.yaml
   ```

4. If required, customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include:
   - Custom Domains and Certificates
   - HTTP proxy
   - Disabling the AI Navigator application
   - External Load Balancer
   - GPU utilization, etc.
   - Rook Ceph customization for Pre-provisioned environments (see page 1541)

5. *If required:* If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

   ```
   ...
   ```

```
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true
...
```

6. Expand **one** of the following sets of instructions, depending on your license and application environments:

Essential License: Install Kommander in a Non-Air-Gapped Environment

6.12.8.7.1.2 Essential License: Install Kommander

Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Enterprise License: Install Kommander in a Non-air-gapped Environment with DKP Catalog Applications

6.12.8.7.1.3 Enterprise License: Install Kommander with DKP Catalog Applications

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
```

```
        ref:
          tag: v2.7.0
...
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

6.12.8.7.2 Kommander Customizations

You can configure the Kommander component of DKP during the initial installation, and also post-installation using the DKP CLI. If you are not sure of what you want to customize during install, then proceed to the next step. To read about Kommander component customization options, refer to this section of the documetation: Kommander Customizations (see page 1558)

6.12.8.7.3 Next Step

Day 2 - Cluster Operations Management (see page 590)

6.12.9 AWS Install in an Air-gapped Environment

This section provides the instructions to install DKP in an AWS air-gapped environment with custom settings. First you create an Air-gapped Bootstrap Cluster (see page 1216) and then move the CAPI resources to the workload cluster and delete the bootstrap cluster.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

6.12.9.1 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

1. Follow the steps to create permissions and roles on the Minimal Permissions and Role to Create Clusters (see page 1156) page.
2. Create Cluster IAM Policies and Roles (see page 1162).
3. Export the AWS region where you want to deploy the cluster:

```
export AWS_REGION=us-west-2
```

4. Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

```
export AWS_PROFILE=<profile>
```

If using AWS ECR as your local private registry, more information can be found on the Registry Mirror Tools (see page 1606) page.

To deploy a cluster with a custom image in a region where CAPI images[866] are not provided, you need to use Konvoy Image Builder (see page 1153) to create your own image for the region.

For multi-tenancy, every tenant should be in a different AWS account to ensure they are truly independent of other tenants in order to enforce security.

For air-gapped, ensure you have downloaded (see page 76) `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz` , so you can extract the tarball on next page.

6.12.9.2 Next Step

Custom AMI in Air-gapped Cluster Creation (see page 1208)

6.12.9.3 Custom AMI in Air-gapped Cluster Creation

6.12.9.3.1 Air-gapped AMI

To create an Image using Konvoy Image Builder (KIB) for use in an air-gapped cluster, follow these instructions. AMI images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create an AMI image of your current computer system settings and software. The AMI image can then be replicated and distributed, creating your computer system for

866 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html

other users. You can use overrides files to customize some of the components installed on your machine image. For example, you could tell KIB to install the FIPS versions of the Kubernetes components.

6.12.9.3.2 Prerequisites

- Minimal IAM Permissions for KIB (see page 1629) to create an Image for an AWS account using Konvoy Image Builder.

> ⚠ In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1626).
> Explore the Customize your Image (see page 1661) topic for more options.
> Using KIB (see page 1626), you can build an AMI without requiring access to the internet by providing an additional `--override` flag.

In previous DKP releases, the distro package bundles were included in the downloaded air-gapped bundle. Currently, that air-gapped bundle contains the following artifacts with the exception of the distro packages:

- DKP Kubernetes packages
- Python packages (provided by upstream)
- Containerd tarball

1. Download (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz && cd dkp-v2.8.1/kib
   ```

2. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.
3. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle` . This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml` . Example command:

   ```
   ./konvoy-image create-package-bundle --os redhat-8.4 --output-
   directory=artifacts
   ```

   NOTE: For FIPS, pass the flag: `--fips`

   NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY` . Example command:

```
export RHSM_ACTIVATION_KEY="-ci"
export RHSM_ORG_ID="1232131"
```

4. Follow the instructions below to build an AMI.

Depending on which version of DKP (see page 1626) you are running, steps and flags will be different. To deploy in a region where CAPI images are not provided, you need to use KIB to create your own image for the region. For a list of supported AWS regions, refer to the Published AMI[867]information from AWS.

6.12.9.3.2.1 Run the following to begin image creation:

5. Run the `konvoy-image` command to build and validate the image. Ensure you have named the correct AMI image YAML file for your OS[868] in the `konvoy-image build` command.

```
./konvoy-image build aws images/ami/centos-79.yaml --overrides overrides/
offline.yaml
```

By default, it builds in the `us-west-2` region. **To specify another region** set the `--region` flag as shown in the example command below:

```
./konvoy-image build aws --region us-east-1 images/ami/centos-79.yaml --overrides
overrides/offline.yaml
```

Instructions for customizing an override file are found on this page: Image Overrides (see page 1678)

After KIB provisions the image successfully, the `ami` id is printed and written to the `packer.pkr.hcl` (Packer config) file. This file has an `amazon-ebs.kib_image` field whose value provides the name of the AMI ID as shown in the example below. That is the `ami` you use in the `dkp create cluster` command:

```
...
    amazon-ebs.kib_image: Adding tag: "distribution_version": "8.6"
    amazon-ebs.kib_image: Adding tag: "gpu_nvidia_version": ""
    amazon-ebs.kib_image: Adding tag: "kubernetes_cni_version": ""
    amazon-ebs.kib_image: Adding tag: "build_timestamp": "20231023182049"
    amazon-ebs.kib_image: Adding tag: "gpu_types": ""
    amazon-ebs.kib_image: Adding tag: "kubernetes_version": "1.28.7"
```

867 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html
868 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ami

```
==> amazon-ebs.kib_image: Creating snapshot tags
    amazon-ebs.kib_image: Adding tag: "ami_name": "konvoy-ami-
rhel-8.6-1.26.6-20231023182049"
==> amazon-ebs.kib_image: Terminating the source AWS instance...
==> amazon-ebs.kib_image: Cleaning up any extra volumes...
==> amazon-ebs.kib_image: No volumes to clean up, skipping
==> amazon-ebs.kib_image: Deleting temporary security group...
==> amazon-ebs.kib_image: Deleting temporary keypair...
==> amazon-ebs.kib_image: Running post-processor:  (type manifest)
Build 'amazon-ebs.kib_image' finished after 26 minutes 52 seconds.

==> Wait completed after 26 minutes 52 seconds

==> Builds finished. The artifacts of successful builds are:
--> amazon-ebs.kib_image: AMIs were created:
us-west-2: ami-04b8dfef8bd33a016

--> amazon-ebs.kib_image: AMIs were created:
us-west-2: ami-04b8dfef8bd33a016
```

6.12.9.3.2.2 Use the Custom AMI in Cluster Creation

To use a custom AMI when creating your cluster (see page 1190), you must create that AMI using KIB first. Then perform the export and name the custom AMI for use in the command `dkp create cluster`:

The export command is similar to this:

```
export AWS_AMI_ID=ami-ami-04b8dfef8bd33a016
```

The create cluster command would then be:

```
dkp create cluster aws --ami=${AWS_AMI_ID}
```

Inside the section for Air-gapped (see page 1218) cluster creation, you will find the instructions for how to apply custom images.

More information about using the custom AMI in cluster creation can be found here: Use Custom AMI to Build Cluster (see page 1640).

6.12.9.3.2.3 Next Step

AWS Air-gapped Loading the Registry (see page 1211)

6.12.9.4 AWS Air-gapped Loading the Registry

After you create an image for your air-gapped environment, you will need to load it into your registry.

6.12.9.4.1 Load Images into your Registry

Because air-gapped environments do not have direct access to the Internet, you must download, extract and load several required images to your local container registry (see page 1606), before installing DKP.

6.12.9.4.2 Download all Images for Air-gapped Deployments

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for prerequisites to download and then how to push the necessary images to this registry.

1. Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`) to load registry images as explained below.
2. Connectivity with clusters attaching to the management cluster is required:
   - Both management and attached clusters must be able to connect to the local registry.
   - The management cluster must be able to connect to all attached cluster's API servers.
   - The management cluster must be able to connect to any load balancers created for platform services on the management cluster.

6.12.9.4.3 Extract Air-gapped Images and Set Variables

Follow these steps to **extract** the air-gapped image bundles into your private registry using these examples for ECR:

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. For the bootstrap, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

   ```
   cd dkp-v2.8.1
   ```

3. Set an environment variable with your registry address for **ECR**:

   ```
   export REGISTRY_URL=<ecr-registry-URI>
   ```

NOTE: To use ECR:

```
export REGISTRY_URL=<ecr-registry-URI>
```

- `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- The environment where you are running the `dkp push` command must be authenticated with AWS in order to load your images into ECR.

More Registry Variables:

For other registries, more environment variables would be:

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
export REGISTRY_CA=<path to the cacert file on the bastion>
```

6.12.9.4.3.1 Load Images to your Private Registry - Konvoy

Before creating or upgrading a Kubernetes cluster, you need to load the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, refer to Local Registry Tools (see page 1606) page for more information.

Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

For specific `push` flags, refer to the dkp push bundle (see page 1910)section of CLI commands.

Additional Flags for Registry push:

The `push` command will be different depending on username and password requirements:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.7.0.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

6.12.9.4.3.2 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL}
```

Additional Flags for Registry push:

The `push` command will be different depending on username and password requirements:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.7.0.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

6.12.9.4.3.3 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL}
```

Additional Flags for Registry push:

The `push` command will be different depending on username and password requirements:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.7.0.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME}
--to-registry-password=${REGISTRY_PASSWORD}
```

6.12.9.4.3.4 Related Topic

AWS Air-gapped Environment Variables (see page 1215)

6.12.9.4.3.5 Next Step

AWS Air-gapped Bootstrap (see page 1216)

6.12.9.4.4 AWS Air-gapped Environment Variables

When using an existing air-gapped infrastructure, DKP does **not** create, modify, or delete any AWS resources.

AWS Resources:

- Internet Gateways
- NAT Gateways
- Routing tables
- Subnets
- VPC
- VPC Endpoints (for subnets without NAT Gateways)

> An AWS subnet has Network ACLs that can control traffic in and out of the subnet. DKP does not modify the Network ACLs of an existing subnet. DKP uses Security Groups to control traffic. If a Network ACL denies traffic that is allowed by DKP-managed Security Groups, the cluster may not work correctly.

Export variables for the existing infrastructure details:

```
export AWS_VPC_ID=<vpc-...>
export AWS_SUBNET_IDS=<subnet-...,subnet-...,subnet-...>
export AWS_ADDITIONAL_SECURITY_GROUPS=<sg-...>
export AWS_AMI_ID=<ami-...>
```

- `AWS_VPC_ID` : the VPC ID where the cluster will be created. The VPC requires the following AWS VPC Endpoints to be already present:
  - `ec2` - `com.amazonaws.{region}.ec2`
  - `elasticloadbalancing` - `com.amazonaws.{region}.elasticloadbalancing`
  - `secretsmanager` - `com.amazonaws.{region}.secretsmanager`
  - `autoscaling` - `com.amazonaws.{region}.autoscaling`
  - `ecr` - `com.amazonaws.{region}.ecr.api` - `(authentication)`

    - `ecr` - `com.amazonaws.{region}.ecr.dkr` - `(data trasfer)`

  More details about AWS service using an interface VPC endpoint[869] and AWS VPC endpoints list[870].
- `AWS_SUBNET_IDS` : a comma-separated list of one or more private Subnet IDs with each one in a different Availability Zone. The cluster control-plane and worker nodes will automatically be spread across these Subnets.
- `AWS_ADDITIONAL_SECURITY_GROUPS` : a comma-seperated list of one or more Security Groups IDs to use in addition to the ones automatically created by CAPA[871].
- `AWS_AMI_ID` : the AMI ID to use for control-plane and worker nodes. The default AWS image is not recommended for use in production. The AMI must be created with Konvoy Image Builder (see page 1626) in order to use the registry mirror feature.
    - Flag to use custom AMI during cluster creation: `--ami=${AWS_AMI_ID}`

⚠️ **IMPORTANT:** You must tag the subnets as described below to allow for Kubernetes to create ELBs for services of type `LoadBalancer` in those subnets. If the subnets are not tagged, they will not receive an ELB and the following error displays: `Error syncing load balancer, failed to ensure load balancer; could not find any suitable subnets for creating the ELB.`.

The tags should be set as follows, where `<CLUSTER_NAME>` corresponds to the name set in `CLUSTER_NAME` environment variable:

```
kubernetes.io/cluster = <CLUSTER_NAME>
kubernetes.io/cluster/<CLUSTER_NAME> = owned
kubernetes.io/role/internal-elb = 1
```

6.12.9.4.4.1 Next Step

AWS Air-gapped Bootstrap (see page 1216)

6.12.9.5 AWS Air-gapped Bootstrap

6.12.9.5.1 Bootstrap a kind cluster and CAPI controllers

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, after which the workload cluster manages its own lifecycle.

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz` , extract the tarball to a local directory:

869 https://docs.aws.amazon.com/vpc/latest/privatelink/create-interface-endpoint.html
870 https://docs.aws.amazon.com/vpc/latest/privatelink/aws-services-privatelink-support.html
871 https://github.com/kubernetes-sigs/cluster-api-provider-aws

```
tar -xzvf dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz && cd dkp-v2.8.0
```

2. Set an environment variable with your registry address with this command:

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_URL=<ecr-registry-URI>
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
export REGISTRY_CA=<path to the cacert file on the bastion>
```

`REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.

For ECR, the environment where you are running the `dkp push` command must be authenticated with AWS in order to load your images into ECR.

3. Seed the registry by running the following command to load the air-gapped image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.0.tar --
to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-
registry-password=${REGISTRY_PASSWORD}
```

4. Load the bootstrap container image on your bastion machine using Docker or Podman command:

```
docker load -i konvoy-bootstrap-image-v2.8.0.tar
```

```
podman load -i konvoy-bootstrap-image-v2.8.0.tar
```

5. Create a bootstrap cluster:

```
dkp create bootstrap --kubeconfig $HOME/.kube/config
```

Configuring an HTTP/HTTPS Proxy (see page 1053) use `--http-proxy` , `--https-proxy` , and `--no-proxy` and their related values in this command for it to be successful.

HTTP Proxy flags if needed:

To create a bootstrap cluster in a proxied environment (see page 1054) use this command syntax, in addition to any other flags you may need:

```
--http-proxy <string> \
--https-proxy <string> \
--no-proxy <string>
```

6. (Optional) Refresh the credentials used by the AWS provider at any time, using the command:

```
dkp update bootstrap credentials aws
```

Konvoy creates a bootstrap cluster using KIND[872] as a library. Konvoy then deploys the following Cluster API[873] providers on the cluster:

- Core Provider[874]
- vSphere Infrastructure Provider[875]
- Kubeadm Bootstrap Provider[876]
- Kubeadm ControlPlane Provider[877]

6.12.9.5.2 Next Step

AWS Create a New Air-gapped Cluster (see page 1218)

6.12.9.6 AWS Create a New Air-gapped Cluster

6.12.9.6.1 Prerequisites

Before you begin, make sure you have created a Bootstrap (see page 1216) cluster.

> By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

872 https://github.com/kubernetes-sigs/kind
873 https://cluster-api.sigs.k8s.io/
874 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/
875 https://github.com/kubernetes-sigs/cluster-api-provider-vsphere
876 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/bootstrap/kubeadm
877 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/controlplane/kubeadm

6.12.9.6.1.1 Create a New AWS Air-gapped Kubernetes Cluster

Create a new AWS Kubernetes cluster

1. Name your cluster using the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. In AWS it is critical that the name is unique, as no two clusters in the same AWS account can have the same name. See Kubernetes[878] for more naming information.
   Set the environment variable to the name you assigned this cluster:

   ```
   export CLUSTER_NAME=<aws-example>
   ```

2. Export the variables such as custom AMI and existing infrastructure details for later use with the `dkp create cluster` command:
   See the Custom AMI in Cluster Creation (see page 1185) topic for more information.

   ```
   export AWS_AMI_ID=<ami-...>
   export AWS_VPC_ID=<vpc-...>
   export AWS_SUBNET_IDS=<subnet-...,subnet-...,subnet-...>
   export AWS_ADDITIONAL_SECURITY_GROUPS=<sg-...>
   ```

Flags to use during cluster creation to apply these variables:

```
--ami=${AWS_AMI_ID} \
--vpc-id=${AWS_VPC_ID} \
--subnet-ids=${AWS_SUBNET_IDS} \
--additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
```

In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1626).

There are two approaches to supplying the ID of your AMI. Either provide the ID of the AMI or provide a way for DKP to discover the AMI using location, format and OS information:

- **Option One** - Provide the ID of your AMI:
  - Use the example command below leaving the existing flag that provides the AMI ID: `--ami AMI_ID`
- **Option Two** - Provide a path for your AMI with the information required for image discover:

878 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

a. Where the AMI is published using your AWS Account ID: `--ami-owner AWS_ACCOUNT_ID`
b. The format or string used to search for matching AMIs and ensure it references the Kubernetes version plus the base OS name: `--ami-base-os ubuntu-20.04`
c. The base OS information: `--ami-format 'example-{{.BaseOS}}-?{{.K8sVersion}}-*'`

When using an existing air-gapped infrastructure, DKP does **not** create, modify, or delete the AWS resources such as Internet gateways, VPSs, subnets, etc. For more information, refer to AWS Air-gapped Environment Variables (see page 1215)

3. Ensure your subnets (see page 1065) do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

4. Your cluster uses the existing local registry (see page 1606) when attempting to pull images previously pushed to your registry (see page 1211). The registry variables were set during the BOOTSTRAP (see page 1216) process on previous page and will be used with the flags during the `dkp create cluster` command next.

Registry flags to use during cluster creation:

```
--registry-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD}
```

See also: Use a Registry Mirror (see page 1609)

Definitions of the variables is still needed:

- `REGISTRY_URL`: the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.

Other local registries may use the options below:

- JFrog - `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional-set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.

To increase Docker Hub's rate limit[879] use your Docker Hub credentials when creating the cluster, by setting flags `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username=<your-username> --registry-mirror-password=<your-password>` when running `dkp create cluster` .

4. Create the Kubernetes cluster objects with a `dry run` . The following example shows a common configuration. See dkp create cluster aws (see page 1856) reference for the full list of cluster creation options:

```
dkp create cluster aws
  --cluster-name=${CLUSTER_NAME} \
  --vpc-id=${AWS_VPC_ID} \
  --ami=${AWS_AMI_ID} \
  --subnet-ids=${AWS_SUBNET_IDS} \
  --internal-load-balancer=true \
  --additional-security-group-ids=${AWS_ADDITIONAL_SECURITY_GROUPS} \
  --registry-mirror-url=${REGISTRY_URL} \
  --dry-run \
  --output=yaml \
  > ${CLUSTER_NAME}.yaml
```

Expand the drop-downs for more flags for use in cluster creation such as **registry, HTTP, FIPS and other flags** to apply in the step above.

Flatcar

Flatcar OS (see page 1183) use this flag used to instruct the bootstrap cluster to make some changes related to the installation paths:

```
--os-hint flatcar
```

HTTP ONLY

879 https://docs.docker.com/docker-hub/download-rate-limit/

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

```
--http-proxy <<http proxy list>>
--https-proxy <<https proxy list>>
--no-proxy <<no proxy list>>
```

- To configure the Control Plane and Worker nodes to use an HTTP proxy:

```
export CONTROL_PLANE_HTTP_PROXY=http://example.org:8080
export CONTROL_PLANE_HTTPS_PROXY=http://example.org:8080
export
CONTROL_PLANE_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1
,10.96.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.s
vc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.sv
c.cluster,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"

export WORKER_HTTP_PROXY=http://example.org:8080
export WORKER_HTTPS_PROXY=http://example.org:8080
export
WORKER_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96.
0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kube
rnetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.clust
er,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"
```

- Replace:
  - `example.org,example.com,example.net` with you internal addresses
  - `localhost` and `127.0.0.1` addresses should not use the proxy
  - `10.96.0.0/12` is the default Kubernetes service subnet
  - `192.168.0.0/16` is the default Kubernetes pod subnet
  - `kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local` is the internal Kubernetes kube-apiserver service
  - `.svc,.svc.cluster,.svc.cluster.local` is the internal Kubernetes services
  - `169.254.169.254` is the AWS metadata server
  - `.elb.amazonaws.com` is for the worker nodes to allow them to communicate directly to the kube-apiserver ELB
- Apply the HTTP flags during the `dkp create cluster` command:

```
--control-plane-http-proxy=${CONTROL_PLANE_HTTP_PROXY} \
```

```
--control-plane-https-proxy=${CONTROL_PLANE_HTTPS_PROXY} \
--control-plane-no-proxy=${CONTROL_PLANE_NO_PROXY} \
--worker-http-proxy=${WORKER_HTTP_PROXY} \
--worker-https-proxy=${WORKER_HTTPS_PROXY} \
--worker-no-proxy=${WORKER_NO_PROXY}
```

FIPS considerations

To create a cluster in FIPS mode (see page 1598), inform the controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes by adding those flags to `dkp create cluster` command:

```
--kubernetes-version=v1.28.7+fips.0 \
--etcd-version=3.5.10+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
```

Individual manifests using the Output Directory flag:

You can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag. This will create multiple files in the specified directory which must already exist:

```
--output-directory=<existing-directory>
```

Refer to the Cluster Creation Customization Choices (see page 1170) section for more information on how to use optional flags such as the `--output-directory` flag.

5. Inspect or edit the generated cluster objects. Familiarize yourself with Cluster API before editing the cluster objects as edits can prevent the cluster from deploying successfully. See AWS Customizing CAPI Clusters (see page 1173).

   ```
   kubectl get clusters,kubeadmcontrolplanes,machinedeployments
   ```

6. **(Optional)** Modify Control Plane Audit logs - Users can make modifications to the `KubeadmControlplane` cluster-api object to configure different `kubelet` options. See the following guide (see page 1613) if you wish to configure your control plane beyond the existing options that are available from flags.

7. Create the cluster from the objects generated in the `dry run`. A warning will appear in the console if the resource already exists and will require you to remove the resource or update your YAML.

   ```
   kubectl create -f ${CLUSTER_NAME}.yaml
   ```

   NOTE: If you used the `--output-directory` flag in your `dkp create .. --dry-run` step above, create the cluster from the objects you created by specifying the directory:

```
kubectl create -f <existing-directory>/
```

OUTPUT will be similar to output shown in this drop-down:

```
cluster.cluster.x-k8s.io/aws-example created
awscluster.infrastructure.cluster.x-k8s.io/aws-example created
kubeadmcontrolplane.controlplane.cluster.x-k8s.io/aws-example-control-plane created
awsmachinetemplate.infrastructure.cluster.x-k8s.io/aws-example-control-plane created
secret/aws-example-etcd-encryption-config created
machinedeployment.cluster.x-k8s.io/aws-example-md-0 created
awsmachinetemplate.infrastructure.cluster.x-k8s.io/aws-example-md-0 created
kubeadmconfigtemplate.bootstrap.cluster.x-k8s.io/aws-example-md-0 created
clusterresourceset.addons.cluster.x-k8s.io/calico-cni-installation-aws-example
created
configmap/calico-cni-installation-aws-example created
configmap/tigera-operator-aws-example created
clusterresourceset.addons.cluster.x-k8s.io/aws-ebs-csi-aws-example created
configmap/aws-ebs-csi-aws-example created
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-aws-example created
configmap/cluster-autoscaler-aws-example created
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-aws-example created
configmap/node-feature-discovery-aws-example created
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-aws-example
created
configmap/nvidia-feature-discovery-aws-example created
```

8. Wait for the cluster control-plane to be ready:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=60m
```

9. After the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. Konvoy provides a command to describe the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

OUTPUT:

```
NAME                                                          READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/aws-example                                           True
60s
├─ClusterInfrastructure - AWSCluster/aws-example              True
5m23s
```

```
├─ControlPlane - KubeadmControlPlane/aws-example-control-plane  True
60s
│ ├─Machine/aws-example-control-plane-55jh4                     True
4m59s
│ ├─Machine/aws-example-control-plane-6sn97                     True
2m49s
│ └─Machine/aws-example-control-plane-nx9v5                     True
66s
└─Workers
  └─MachineDeployment/aws-example-md-0                          True
117s
    ├─Machine/aws-example-md-0-cb9c9bbf7-hcl8z                  True
3m1s
    ├─Machine/aws-example-md-0-cb9c9bbf7-rtdqw                  True
3m2s
    ├─Machine/aws-example-md-0-cb9c9bbf7-t894m                  True
3m1s
    └─Machine/aws-example-md-0-cb9c9bbf7-td29r                  True
```

10. As they progress, the controllers also create Events. List the Events using this command:

```
kubectl get events | grep ${CLUSTER_NAME}
```


DKP uses AWS CSI as the default storage provider (see page 110). You can use a Kubernetes CSI[880] compatible storage solution that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[881] for more information.
If you're not using the default, you cannot deploy an alternate provider until ***after*** the `dkp create cluster` is finished. However, it must be determined before Kommander installation.

6.12.9.6.1.2 Next Step

Make the AWS Air-gapped Cluster Self-Managed (see page 1227)

6.12.9.6.2 Option for Creating DKP Clusters on AWS from the UI

A guide for creating DKP clusters on AWS console

880 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
881 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

6.12.9.6.2.1 Before you Begin

Configure an AWS Infrastructure (see page 1148), or add credentials using AWS role credentials (see page 625).

6.12.9.6.2.2 Simplified Cluster Creation on AWS in the UI

1. In the selected workspace Dashboard, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
2. On the Add Cluster page, select the **Create DKP Cluster**.
3. Provide some basic cluster details within the form:
    - **Workspace**: The workspace where this cluster belongs (if within the Global workspace).
    - **Kubernetes Version**: The initial version of Kubernetes to install on the cluster.
    - **Name**: A valid Kubernetes name for the cluster.
    - **Add Labels**: By default, your cluster has labels that reflect the infrastructure provider provisioning. For example, your AWS cluster may have a label for the data center region and `provider: aws`. Cluster labels are matched to the selectors created for Projects (see page 716). Changing a cluster label may add or remove the cluster from projects.
4. Select the pre-configured AWS Infrastructure (see page 1148), or AWS role credentials (see page 625) to display the remaining options specific to AWS.
    - **Region**: Select a data center region or specify a custom region.
    - **Configure Node Pools**: Specify pools of nodes, their machine types, quantity, and the IAM instance profile.
        - **Machine Type**: Machine instance type.
        - **Quantity**: Number of nodes. The control plane must be an odd number.
        - **IAM instance profile**: Name of the IAM instance profile to assign to the machines.
    - **AMI Image**: Specify the AMI to use for each node pool using either method (a) OR method (b) below:
        - (a) AMI ID specifying by name.
            - **AMI ID**: AMI ID to use for all nodes.
        - (b) AMI ID for lookup - Owner ID, Base OS, Lookup Format (all fields are required, if any is used)
            - **Base OS**: Base OS for Lookup search.
            - **Lookup Format**: Lookup Format string to generate AMI search name from.
            - **Owner ID**: Owner ID for AMI Lookup search.
    - **Worker Availability Zone:** Worker Zones for a region (worker nodes only).

- **Add Infrastructure Provider Tags**: Specify tags applied on all resources created in your infrastructure for this cluster. Different infrastructure providers have varying restrictions on the usable tags. See the AWS Tags User Guide[882] for more information on using tags in AWS.

5. Select **Create** to begin provisioning the cluster. This step may take a few minutes, taking time for the cluster to be ready and fully deploy its components. The cluster automatically tries to join, and should resolve after it is fully-provisioned.

6.12.9.7 Make the AWS Air-gapped Cluster Self-Managed

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which then deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, which makes the workload cluster self-managed. This section describes how to make a workload cluster self-managed.

This page contains instructions on how to make your cluster self-managed. This is necessary if there is only one cluster in your environment, or if this cluster should become the Management cluster in a multi-cluster environment.

If you already have a self-managed or Management cluster in your environment, skip this page.

6.12.9.7.1 Make the New Kubernetes Cluster Manage Itself

1. Deploy cluster lifecycle services on the workload cluster:

```
dkp create capi-components --kubeconfig ${CLUSTER_NAME}.conf
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

```
✓ Initializing new CAPI components
```

2. Move the Cluster API objects from the bootstrap to the workload cluster:

   The cluster lifecycle services on the workload cluster are ready, but the workload cluster configuration is on the bootstrap cluster. The `move` command moves the configuration, which takes

882 https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Using_Tags.html

the form of Cluster API Custom Resource objects, from the bootstrap to the workload cluster. This process is also called a Pivot[883].

```
dkp move capi-resources --to-kubeconfig ${CLUSTER_NAME}.conf
```

```
 ✓ Moving cluster resources
You can now view resources in the moved cluster by using the --kubeconfig flag
with kubectl. For example: kubectl --kubeconfig=aws-example.conf get nodes
```

NOTE: To ensure only one set of cluster lifecycle services manages the workload cluster, Konvoy first pauses reconciliation of the objects on the bootstrap cluster, then creates the objects on the workload cluster. As Konvoy copies the objects, the cluster lifecycle services on the workload cluster reconcile the objects. The workload cluster becomes self-managed after Konvoy creates all the objects. If it fails, the `move` command can be safely retried.

3. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf wait --for=condition=ControlPlaneRead
y "clusters/${CLUSTER_NAME}" --timeout=20m
```

```
cluster.cluster.x-k8s.io/aws-example condition met
```

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use Konvoy with the workload cluster kubeconfig.

```
dkp describe cluster --kubeconfig ${CLUSTER_NAME}.conf -c ${CLUSTER_NAME}
```

```
NAME                                                           READY
SEVERITY  REASON  SINCE  MESSAGE
```

883 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
Cluster/aws-example                                            True
109s
├─ClusterInfrastructure - AWSCluster/aws-example               True
112s
├─ControlPlane - KubeadmControlPlane/aws-example-control-plane  True
109s
│ ├─Machine/aws-example-control-plane-55jh4                    True
111s
│ ├─Machine/aws-example-control-plane-6sn97                    True
111s
│ └─Machine/aws-example-control-plane-nx9v5                    True
110s
└─Workers
  └─MachineDeployment/aws-example-md-0                         True
114s
    ├─Machine/aws-example-md-0-cb9c9bbf7-hcl8z                 True
111s
    ├─Machine/aws-example-md-0-cb9c9bbf7-rtdqw                 True
111s
    ├─Machine/aws-example-md-0-cb9c9bbf7-t894m                 True
111s
    └─Machine/aws-example-md-0-cb9c9bbf7-td29r                 True
111s
```

5. Remove the bootstrap cluster, as the workload cluster is now self-managed:

```
dkp delete bootstrap
```

```
✓ Deleting bootstrap cluster
```

6.12.9.7.2 Known Limitations

- Before making a workload cluster self-managed, be sure that its control plane nodes have sufficient permissions for running Cluster API controllers. See IAM Policy Configuration (see page 1162).
- Konvoy supports moving only one set of cluster objects from the bootstrap cluster to the workload cluster, or vice-versa.
- Konvoy only supports moving all namespaces in the cluster; Konvoy does not support migration of individual namespaces.

6.12.9.7.3 Next Step:

Explore the New AWS Air-gapped Cluster (see page 1230)

6.12.9.8 Explore the New AWS Air-gapped Cluster

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster.

Follow these steps:

1. Fetch the kubeconfig file with the command:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

2. List the Nodes with the command:

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
   ```

   Note: wait for the Status to move to `Ready` while `calico-node` pods are being deployed.

3. List the Pods with the command:

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get pods -A
   ```

6.12.9.8.1 Next Step:

AWS Install Kommander in an Air-gapped Environment (see page 1230)

6.12.9.9 AWS Install Kommander in an Air-gapped Environment

This section provides installation instructions for the Kommander component of DKP in an air-gapped environment for AWS.

6.12.9.9.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry: Air-gapped Environments (see page 1536).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

6.12.9.9.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

```
export CLUSTER_NAME=<your-management-cluster-name>
```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1558) for the deployment:

```
dkp install kommander --init --airgapped > kommander.yaml
```

4. *If required*, **customize** your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include:
   - Custom Domains and Certificates
   - HTTP proxy
   - External Load Balancer
   - GPU utilization, etc.
   - Rook Ceph customization for Pre-provisioned environments (see page 1541)
5. *If required*: If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
...
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true
...
```

6. Expand **one** of the following sets of instructions, depending on your license and application environments:

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

Essential License: Install Kommander in an Air-Gapped Environment

6.12.9.9.1.2 Essential License: Install Kommander in an Air-gapped Environment

Use the customized `kommander.yaml` to install DKP in an air-gapped environment:

```
dkp install kommander \
--installer-config kommander.yaml --kubeconfig=${CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.0.tar.gz
```

Enterprise License: Install Kommander in an Air-gapped Environment with DKP Catalog Applications

6.12.9.9.1.3 Enterprise License: Install Kommander in an Air-gapped Environment with DKP Catalog Applications

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add the following values to enable DKP Catalog Applications:

   ```
   ...
   catalog:
     repositories:
       - name: dkp-catalog-applications
         labels:
           kommander.d2iq.io/project-default-catalog-repository: "true"
           kommander.d2iq.io/workspace-default-catalog-repository: "true"
           kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
         path: ./dkp-catalog-applications-v2.8.0.tar.gz
   ...
   ```

   ⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

   ```
   dkp install kommander \
   --installer-config kommander.yaml --kubeconfig=${CLUSTER_NAME}.conf \
   ```

```
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.0.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.8.0.tar.gz
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

6.12.9.9.2 Kommander Customizations

You can configure the Kommander component of DKP during the initial installation, and also post-installation using the DKP CLI. If you are not sure of what you want to customize during install, then proceed to the next step. To read about Kommander component customization options, refer to this section of the documetation: Kommander Customizations (see page 1558)

6.12.9.9.3 Next Step

Day 2 - Cluster Operations Management (see page 590)

6.12.10 AWS Management Tools

After cluster creation and configuration, you can revisit clusters to update and change variables.

6.12.10.1 Delete an AWS Cluster

> A self-managed workload cluster cannot delete itself. If your workload cluster is self-managed, you must first create a bootstrap cluster and move the cluster lifecycle services to it, before deleting the workload cluster.

If you did not make your workload cluster self-managed, as described in Make New Cluster Self-Managed (see page 1227), see the instructions in Delete the workload cluster (see page 1236).

6.12.10.1.1 Create a Bootstrap Cluster and Move CAPI Resources

Follow these steps to create a bootstrap cluster and move CAPI resources:

1. Make sure your AWS credentials are up to date. Refresh the credentials using this command:

   ```
   dkp update bootstrap credentials aws --kubeconfig $HOME/.kube/config
   ```

2. The bootstrap cluster will host the Cluster API controllers that reconcile the cluster objects marked for deletion. Create a bootstrap cluster:

   NOTE: To avoid using the wrong kubeconfig, the following steps use explicit kubeconfig paths and contexts.

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config --with-aws-bootstrap-
   credentials=true
   ```

   ```
   ✓ Creating a bootstrap cluster
   ✓ Initializing new CAPI components
   ```

3. Move the Cluster API objects from the workload to the bootstrap cluster: The cluster lifecycle services on the bootstrap cluster are ready, but the workload cluster configuration is on the workload cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the workload to the bootstrap cluster. This process is also called a Pivot[884].

   ```
   dkp move capi-resources \
   ```

884 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
    --from-kubeconfig ${CLUSTER_NAME}.conf \
    --from-context ${CLUSTER_NAME}-admin@${CLUSTER_NAME} \
    --to-kubeconfig $HOME/.kube/config \
    --to-context kind-konvoy-capi-bootstrapper
```

```
✓ Moving cluster resources
You can now view resources in the moved cluster by using the --kubeconfig flag
with kubectl. For example: kubectl --kubeconfig $HOME/.kube/config get nodes
```

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster's status:

```
dkp describe cluster --kubeconfig $HOME/.kube/config -c ${CLUSTER_NAME}
```

```
NAME                                                          READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/aws-example                                           True
91s
├─ClusterInfrastructure - AWSCluster/aws-example              True
103s
├─ControlPlane - KubeadmControlPlane/aws-example-control-plane  True
91s
│ ├─Machine/aws-example-control-plane-55jh4                   True
102s
│ ├─Machine/aws-example-control-plane-6sn97                   True
102s
│ └─Machine/aws-example-control-plane-nx9v5                   True
102s
└─Workers
  └─MachineDeployment/aws-example-md-0                        True
108s
    ├─Machine/aws-example-md-0-cb9c9bbf7-hcl8z                True
102s
    ├─Machine/aws-example-md-0-cb9c9bbf7-rtdqw                True
102s
    ├─Machine/aws-example-md-0-cb9c9bbf7-td29r                True
102s
    └─Machine/aws-example-md-0-cb9c9bbf7-w64kg                True
102s
```

NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use DKP with the workload cluster kubeconfig and use DKP with the bootstrap cluster to delete the workload cluster.

5. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig $HOME/.kube/config wait --for=condition=controlplaneready
"clusters/${CLUSTER_NAME}" --timeout=20m
```

```
cluster.cluster.x-k8s.io/aws-example condition met
```

6.12.10.1.2 Delete the Workload Cluster

1. Make sure your AWS credentials are up to date. Refresh the credentials using this command:

```
dkp update bootstrap credentials aws --kubeconfig $HOME/.kube/config
```

NOTE: Persistent Volumes (PVs) are not deleted automatically by design to preserve your data. However, the PVs take up storage space if not deleted. You must delete PVs manually. Information for backup of a cluster and PVs is on the page, Back up your Cluster's Applications and Persistent Volumes (see page 863).

2. To delete a cluster, you would use `dkp delete cluster` and pass in the name of the cluster you are trying to delete with `--cluster-name` flag. You would use `kubectl get clusters` to get those details (`--cluster-name` and `--namespace`) of the Kubernetes cluster to delete it.
NOTE: Do not use `dkp get clusters` since that gets you Kommander cluster details rather than Konvoy kubernetes cluster details.

```
kubectl get clusters
```

3. Delete the Kubernetes cluster and wait a few minutes:

NOTE: Before deleting the cluster, dkp deletes all Services of type LoadBalancer on the cluster. Each Service is backed by an AWS Classic ELB. Deleting the Service deletes the ELB that backs it. To skip this step, use the flag `--delete-kubernetes-resources=false` . Do not skip this step if the VPC is managed by DKP. When DKP deletes the cluster, it deletes the VPC. If the VPC has any AWS Classic ELBs, AWS does not allow the VPC to be deleted, and DKP cannot delete the cluster.

```
dkp delete cluster --cluster-name=${CLUSTER_NAME} --kubeconfig $HOME/.kube/
config
```

```
✓ Deleting Services with type LoadBalancer for Cluster default/azure-example
✓ Deleting ClusterResourceSets for Cluster default/azure-example
✓ Deleting cluster resources
✓ Waiting for cluster to be fully deleted
Deleted default/azure-example cluster
```

After the workload cluster is deleted, you can delete the bootstrap cluster.

6.12.10.1.3 Delete the Bootstrap Cluster

After you have moved the workload resources back to a bootstrap cluster and deleted the workload cluster, you no longer need the bootstrap cluster. You can safely delete the bootstrap cluster with these steps:

1. Make sure your AWS credentials are up to date. Refresh the credentials using this command:

   ```
   dkp update bootstrap credentials aws --kubeconfig $HOME/.kube/config
   ```

2. Delete the bootstrap cluster:

   ```
   dkp delete bootstrap --kubeconfig $HOME/.kube/config
   ```

   ```
   ✓ Deleting bootstrap cluster
   ```

6.12.10.1.4 Known Limitations

- The Konvoy version used to create the workload cluster must match the Konvoy version used to delete the workload cluster.

6.12.10.1.5 Next Topic:

Multiple AWS Accounts (see page 1238)

6.12.10.2 Multiple AWS Accounts

Leverage Multiple AWS Accounts for Kubernetes Cluster Deployments

6.12.10.2.1 Objective

You can leverage multiple AWS accounts in your organization to meet specific business purposes, reflect your organizational structure, or implement a multi-tenancy strategy. Specific scenarios include:

- Implementing isolation between environment tiers such as development, testing, acceptance, and production.
- Implementing separation of concerns between management clusters, and workload clusters.
- Reducing the impact of security events and incidents.

For additional benefits of using multiple AWS accounts, refer to the following white paper[885].

This document describes how to leverage the D2iQ Kubernetes Platform (DKP) to deploy a management cluster, and multiple workload clusters, leveraging multiple AWS accounts.

6.12.10.2.2 Assumptions

This guide assumes you have some understanding of Cluster API concepts and basic DKP provisioning workflows on AWS.

Cluster API Concepts - cluster API concepts[886]

AWS Install Options (see page 294)

6.12.10.2.3 Glossary

- **Management cluster** - The cluster that runs in AWS and is used to create target clusters in different AWS accounts.
- **Target account** - The account where the target cluster is created.
- **Source account** - The AWS account where the CAPA controllers for the management cluster runs.

6.12.10.2.4 Prerequisites

Before you begin deploying DKP on AWS, you configure the prerequisites for the environment you use either non-air-gapped or air-gapped.

Prerequisites for Install (see page 141)

885 https://docs.aws.amazon.com/whitepapers/latest/organizing-your-aws-environment/benefits-of-using-multiple-aws-accounts.html
886 https://cluster-api.sigs.k8s.io/user/concepts.html

6.12.10.2.5 Deploy DKP on AWS

1. Deploy a management cluster in your AWS source account.
   AWS: create Kubernetes AWS cluster (see page 1190)
2. Configure a trusted relationship between source and target accounts and create a management cluster:

6.12.10.2.5.1 Step 1:

DKP leverages the Cluster API provider for AWS (CAPA) to provision Kubernetes clusters in a declarative way. Customers declare the desired state of the cluster through a cluster configuration YAML file which is generated using:

(AWS)

```
dkp create cluster aws --cluster-name=${CLUSTER_NAME} \
--dry-run \
--output=yaml \
> ${CLUSTER_NAME}.yaml
```

6.12.10.2.5.2 Step 2:

Configure a trust relationship between the source and target accounts.

Follow all the prerequisite steps in both the source and target accounts

1. Create all policies and roles in management and workload accounts a. The prerequisite IAM policies for DKP are documented here: Configure AWS IAM policies (see page 1162).
2. Establish a trust relationship in workload account for the management account.

   a. Go to your target (workload) account b. Search for the role control-plane.cluster-api-provider-aws.sigs.k8s.io c. Navigate to the Trust Relationship tab and select Edit Trust Relationship d. Add the following relationship:

   ```
   {
     "Effect": "Allow",
     "Principal": {
        "AWS": "arn:aws:iam::${mgmt-aws-account}:role/control-plane.cluster-api-
   provider-aws.sigs.k8s.io"
     },
     "Action": "sts:AssumeRole"
   }
   ```

3. Give permission to role in the source (management cluster) account to call the `sts:AssumeRole API` a. Log in to the source AWS account and attach the following inline policy to control-

plane.cluster-api-provider-aws.sigs.k8s.io role:

```
{
  "Version": "2012-10-17",
  "Statement": [
     {
       "Effect": "Allow",
       "Action": "sts:AssumeRole",
       "Resource": [
"arn:aws:iam::${workload-aws-account}:role/control-plane.cluster-api-provider-
aws.sigs.k8s.io"
       ]
     }
  ]
}
```

4. Modify the management cluster configuration file and update the AWSCluster object with following details:

```
apiVersion: infrastructure.cluster.x-k8s.io/v1alpha3
kind: AWSCluster
metadata:
spec:
  identityRef:
     kind: AWSClusterRoleIdentity
     name: cross-account-role
…
…
---
apiVersion: infrastructure.cluster.x-k8s.io/v1alpha3
kind: AWSClusterRoleIdentity
metadata:
  name: cross-account-role
spec:
  allowedNamespaces: {}
  roleARN: "arn:aws:iam::${workload-aws-account}:role/control-plane.cluster-
api-provider-aws.sigs.k8s.io"
  sourceIdentityRef:
    kind: AWSClusterControllerIdentity
    name: default
```

After performing the above steps, your Management cluster will be configured to create new managed clusters in the target AWS workload account.

6.12.10.2.6 Next Topic

AWS Cluster Autoscaler (see page 1241)

6.12.10.3 AWS Cluster Autoscaler

This page explains how to configure autoscaler for node pools. Cluster Autoscaler[887] provides the ability to automatically scale-up or scale-down the number of worker nodes in a cluster, based on the number of pending pods to be scheduled. Running the Cluster Autoscaler is optional.

Unlike Horizontal-Pod Autoscaler[888], Cluster Autoscaler does not depend on any Metrics server and does not need Prometheus or any other metrics source.

The Cluster Autoscaler looks at the following annotations on a MachineDeployment to determine its scale-up and scale-down ranges:

```
cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size
cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size
```

The full list of command line arguments to the Cluster Autoscaler controller is on the Kubernetes public GitHub repository[889].

For more information about how Cluster Autoscaler works, see these documents:

- What is Cluster Autoscaler[890]
- How does scale-up work[891]
- How does scale-down work[892]
- CAPI Provider for Cluster Autoscaler[893]

6.12.10.3.1 Cluster Autoscaler Prerequisites

Before you begin, you must have:

- A Bootstrap Cluster Lifecycle (see page 1187).
- Created a new Kubernetes Cluster (see page 1190).
- A Self-Managed Cluster (see page 1227).

6.12.10.3.1.1 Run Cluster Autoscaler on the Management Cluster

For DKP Enterprise Management Cluster and Essential Clusters, run the following steps to enable Cluster Autoscaler on a Self-managed cluster (see page 1201):

1. Ensure the Cluster Autoscaler controller is up and running (no restarts and no errors in the logs)

887 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi
888 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-fast-is-hpa-when-combined-with-ca
889 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-are-the-parameters-to-ca
890 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-is-cluster-autoscaler
891 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-up-work
892 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-down-work
893 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf logs deployments/cluster-autoscaler
cluster-autoscaler -n kube-system -f
```

2. Enable Cluster Autoscaler by setting the min & max ranges

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment $
{NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size=2
kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment $
{NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size=6
```

3. The Cluster Autoscaler logs will show that the worker nodes are associated with node-groups and that pending pods are being watched.
4. To demonstrate that it is working properly, create a large deployment which will trigger pending pods (For this example we used AWS m5.2xlarge worker nodes. If you have larger worker-nodes, you should scale up the number of replicas accordingly).

```
cat <<EOF | kubectl --kubeconfig=${CLUSTER_NAME}.conf apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox-deployment
  labels:
    app: busybox
spec:
  replicas: 600
  selector:
    matchLabels:
      app: busybox
  template:
    metadata:
      labels:
        app: busybox
    spec:
      containers:
      - name: busybox
        image: busybox:latest
        command:
          - sleep
          - "3600"
        imagePullPolicy: IfNotPresent
      restartPolicy: Always
EOF
```

5. Cluster Autoscaler will scale up the number of Worker Nodes until there are no pending pods.
6. Scale down the number of replicas for `busybox-deployment`.

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf scale --replicas=30 deployment/
busybox-deployment
```

7. Cluster Autoscaler starts to scale down the number of Worker Nodes after the default timeout of 10 minutes.

6.12.10.3.1.2 Run Cluster Autoscaler on a Managed(Workload) Cluster

Unlike the Management(self-managed) cluster instructions above, to run autoscaler on a managed cluster an additional instance of autoscaler is required. This instance is run on the management cluster, but must be pointed at the managed cluster. The `dkp create cluster` command for building a managed cluster would have to be run against the Management cluster so that the `clusterresourcset` for that cluster's autoscaler is modified to deploy the autoscaler on the management cluster itself. The flags for cluster-autoscaler are changed as well.

1. Create a secret with a kubeconfig file of the master cluster in the managed cluster with limited user permissions to only modify resources for the given cluster.
2. Mount the secret into the cluster-autoscaler deployment.
3. Add the following flag to the cluster-autoscaler command so that `/mnt//masterconfig/value` is the path where the master cluster's kubeconfig is loaded via the secret created.

```
--cloud-config=/mnt//masterconfig/value
```

6.12.10.3.1.3 Next Topic

Manage AWS Node Pools (see page 1243)

6.12.10.4 Manage AWS Node Pools

Node pools are part of a cluster and managed as a group, and you can use a node pool to manage a group of machines using the same common properties. When Konvoy creates a new default cluster, there is one node pool for the worker nodes and all nodes in that new node pool have the same configuration. You can create additional node pools for more specialized hardware or configuration. For example, if you want to tune your memory usage on a cluster where you need maximum memory for some machines and minimal memory on other machines, you would create a new node pool with those specific resource needs.

DKP implements node pools using Cluster API MachineDeployments[894]. For more information on node pools, see these sections:

- Create AWS Node Pools (see page 1244)
- List AWS Node Pools (see page 1245)
- Scale AWS Node Pools (see page 1246)

894 https://cluster-api.sigs.k8s.io/developer/architecture/controllers/machine-deployment.html

- Delete AWS Node Pools (see page 1249)

6.12.10.4.1 Create AWS Node Pools

Creating a node pool is useful when you need to run workloads that require machines with specific resources, such as a GPU, additional memory, or specialized network or storage hardware.

6.12.10.4.1.1 Prepare the Environment

1. Set the environment variable to the name you assigned this cluster.

   ```
   export CLUSTER_NAME=aws-example
   ```

   See AWS Install Options (see page 294) for information on naming your cluster.

2. If your workload cluster is self-managed, as described in Make the New Cluster Self-Managed (see page 1227), configure `kubectl` to use the kubeconfig for the cluster.

   ```
   export KUBECONFIG=${CLUSTER_NAME}.conf
   ```

3. Define your node pool name.

   ```
   export NODEPOOL_NAME=example
   ```

6.12.10.4.1.2 Create an AWS Node Pool

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools is to ensure your cluster has nodes deployed in multiple Availability Zones.

By default, the first Availability Zone in the region will be used for the Nodes in the node pool, set `--availability-zone` to create the Nodes in a different Availability Zone.

Create a new AWS node pool with 3 replicas using this command:

```
dkp create nodepool aws ${NODEPOOL_NAME} \
    --cluster-name=${CLUSTER_NAME} \
```

```
    --replicas=3
```

This example uses default values for brevity. Use flags to define custom instance types, AMIs, and other properties.

Advanced users can use a combination of the `--dry-run` and `--output=yaml` or `--output-directory=<existing-directory>` flags to get a complete set of node pool objects to modify locally or store in version control.

> For more information regarding this flag or others, please refer to the DKP CLI Create (see page 1875) section of the documentation for either cluster or nodepool and select your provider.

6.12.10.4.1.3 Next Topic

List AWS Node Pools (see page 1245)

6.12.10.4.2 List AWS Node Pools

Use this command to list the node pools of a given cluster. This returns specific properties of each node pool so that you can see the name of the MachineDeployments.

6.12.10.4.2.1 Managed Clusters

To list all node pools for a managed cluster, run:

```
dkp get nodepools --cluster-name=${CLUSTER_NAME} --kubeconfig=${CLUSTER_NAME}.conf
```

The expected output is similar to the following example, indicating the desired size of the node pool, the number of replicas ready in the node pool, and the Kubernetes version those nodes are running:

```
NODEPOOL                        DESIRED             READY
KUBERNETES VERSION
example                         3                   3                   v1.28.7
aws-example-md-0                4                   4                   v1.28.7
```

6.12.10.4.2.2 Attached Clusters

Before running the command to list the attached clusters, ensure that you know the following:

- KUBECONFIG (see page 106) for the management cluster (see page 79) - In order to find the KUBECONFIG for a cluster from the UI, refer to this section in the documentation: Access a Managed or Attached Cluster (see page 858)

- The CLUSTER_NAME of the attached cluster
- The NAMESPACE of the attached cluster

To list all node pools for an attached cluster, run:

```
dkp get nodepools --cluster-name=${ATTACHED_CLUSTER_NAME} --kubeconfig=$
{CLUSTER_NAME}.conf -n ${ATTACHED_CLUSTER_NAMESPACE}
```

The expected output is similar to below:

```
NODEPOOL                    DESIRED             READY               KUBERNETES
VERSION
aws-attached-md-0           4                   4                   v1.28.7
```

Next Topic

Scale AWS Node Pools (see page 1246)

6.12.10.4.3 Scale AWS Node Pools

While you can run Cluster Autoscaler (see page 1241), you can also manually scale your node pools up or down when you need more finite control over your environment. For example, if you require 10 machines to run a process, you can manually set the scaling to run those 10 machines only. However, if also using the Cluster Autoscaler, you must stay within your minimum and maximum bounds.

6.12.10.4.3.1 Prerequisite

Environment variables, such as define the node pool name, are set in the Prepare the Environment (see page 1244) section on the previous page. If need, refer to that page to set those variables.

Scaling Up Node Pools

To scale up a node pool in a cluster, run:

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=5 --cluster-name=${CLUSTER_NAME}
```

OR for attached clusters:

```
dkp scale nodepools ${ATTACHED_NODEPOOL_NAME} --replicas=5 --cluster-name=$
{ATTACHED_CLUSTER_NAME} --kubeconfig=${CLUSTER_NAME}.conf -n $
{ATTACHED_CLUSTER_WORKSPACE}
```

Your output should be similar to this example, indicating the scaling is in progress:

```
✓ Scaling node pool example to 5 replicas
```

After a few minutes, you can list the node pools to:

```
dkp get nodepools --cluster-name=${CLUSTER_NAME} --kubeconfig=${CLUSTER_NAME}.conf
```

Your output should be similar to this example, with the number of DESIRED and READY replicas increased to 5:

```
NODEPOOL                          DESIRED              READY
KUBERNETES VERSION
aws-example-md-0                   5                    5                    v1.28.7

aws-attached-md-0                  5                    5                    v1.28.7
```

Scaling Down Node Pools

To scale down a node pool, run:

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=4 --cluster-name=${CLUSTER_NAME}
```

OR for attached clusters:

```
dkp scale nodepools ${ATTACHED_NODEPOOL_NAME} --replicas=4 --cluster-name=$
{ATTACHED_CLUSTER_NAME} --kubeconfig=${CLUSTER_NAME}.conf -n $
{ATTACHED_CLUSTER_WORKSPACE}
```

```
✓ Scaling node pool example to 4 replicas
```

After a few minutes, you can list the node pools using this command:

```
dkp get nodepools --cluster-name=${CLUSTER_NAME} --kubeconfig=${CLUSTER_NAME}.conf
```

Your output should be similar to this example, with the number of DESIRED and READY replicas decreased to 4:

```
NODEPOOL                          DESIRED              READY
KUBERNETES VERSION
example                           4                    4                  v1.28.7
aws-example-md-0                  4                    4                  v1.28.7
```

In a default cluster, the nodes to delete are selected at random. This behavior is controlled by CAPI's delete policy[895]. However, when using the DKP CLI to scale down a node pool, it is also possible to specify the Kubernetes Nodes you want to delete.

To do this, set the flag `--nodes-to-delete` with a list of nodes as below. This adds an annotation `cluster.x-k8s.io/delete-machine=yes` to the matching Machine object that contains `status.NodeRef` with the node names from `--nodes-to-delete`.

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=3 --nodes-to-delete=<> --cluster-
name=${CLUSTER_NAME}
```

```
✓ Scaling node pool example to 3 replicas
```

Scaling Node Pools When Using Cluster Autoscaler

If you configured the cluster autoscaler (see page 1241) for the `demo-cluster-md-0` node pool, the value of `--replicas` must be within the minimum and maximum bounds.

For example, assuming you have these annotations:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment ${NODEPOOL_NAME}
cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size=2
kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment ${NODEPOOL_NAME}
cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size=6
```

Try to scale the node pool to 7 replicas with the command:

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=7 -c demo-cluster
```

Which results in an error similar to:

```
 ✗ Scaling node pool example to 7 replicas
failed to scale nodepool: scaling MachineDeployment is forbidden: desired replicas 7
 is greater than the configured max size annotation cluster.x-k8s.io/cluster-api-
autoscaler-node-group-max-size: 6
```

Similarly, scaling down to a number of replicas less than the configured `min-size` also returns an error.

Next Topic

Delete AWS Node Pools (see page 1249)

---

895 https://github.com/kubernetes-sigs/cluster-api/blob/v0.4.0/api/v1alpha4/machineset_types.go#L85-L105

6.12.10.4.4 Delete AWS Node Pools

Deleting a node pool deletes the Kubernetes nodes and the underlying infrastructure. All nodes will be drained prior to deletion and the pods running on those nodes will be rescheduled.

To delete a node pool from a managed cluster, run:

```
dkp delete nodepool ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME}
```

Here, `example` is the node pool to be deleted.

The expected output will be similar to the following example, indicating the node pool is being deleted:

```
✓ Deleting default/example nodepool resources
```

Deleting an invalid node pool results in output similar to this example:

```
dkp delete nodepool ${CLUSTER_NAME}-md-invalid --cluster-name=${CLUSTER_NAME}
```

```
MachineDeployments or MachinePools.infrastructure.cluster.x-k8s.io "no
MachineDeployments or MachinePools found for cluster aws-example" not found
```

6.12.10.5 GPUs in an AWS environment

6.12.10.5.1 Understanding GPUs

Before working with GPUs, ensure you are familiar with the following:

- Install GPU support on supported distributions on AWS
- GPU node labeling specifications described in Configure Konvoy Automatic GPU Node Labels (see page 1251)

Kommander also accesses Kommander GPU configuration (see page 1005) resources.

For using GPUs in an air-gapped on-premises environment, D2iQ recommends setting up Pod Disruption Budget[896] before Update Cluster Nodepools (see page 1620).

896 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/

6.12.10.5.2 Install GPU Support for Supported Distributions on AWS

Using the Konvoy Image Builder (see page 1626), you can build an image that has support to use NVIDIA GPU hardware to support GPU workloads. DKP supported NVIDIA driver[897] version is 470.x.

1. In your `overrides/nvidia.yaml` file add the following to enable GPU builds. You can also access and use the overrides repo.[898]

   ```
   gpu:
       type:
       - nvidia
   ```

   Build your image using the following Konvoy image builder commands:

   ```
   konvoy-image build --region us-west-2 --source-ami=ami-12345abcdef images/ami/
   centos-7.yaml --overrides overrides/nvidia.yaml
   ```

   By default, your image builds in the `us-west-2` region. To specify another region, set the `--region` flag:

   ```
   konvoy-image build --region us-east-1 --overrides override-source-ami.yaml
   images/ami/<Your OS>.yaml
   ```

Ensure that an AMI file is available for your OS selection.

1. When the command is complete the `ami` id is printed and written to `packer.pkr.hcl`.

   To use the built `ami` with Konvoy Image Builder, specify it with the `--ami` flag when calling cluster create.

   ```
   dkp create cluster aws --cluster-name=$(whoami)-aws-cluster --region us-west-2
    --ami <ami>
   ```

897 https://www.nvidia.com/Download/Find.aspx
898 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

> If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

Additional helpful information can be found in the NVIDIA Device Plug-in[899] for Kubernetes instructions and the Installation Guide of Supported Platforms[900].

See also: NVIDIA documentation[901]

6.12.10.5.3 Next Topic

Configure Konvoy Automatic GPU Node Labels (see page 1251)

6.12.10.5.4 Configure Konvoy Automatic GPU Node Labels

When using GPU nodes, it is important they have the proper label identifying them as Nvidia GPU nodes. Node feature discovery (NFD), by default labels PCI hardware as:

```
"feature.node.kubernetes.io/pci-<device label>.present": "true"
```

where `<device label>` is by default as defined in this topic[902]:

```
< class > _ < vendor >
```

However, because there is a wide variety in devices and their assigned PCI classes, you may find that the labels assigned to your GPU nodes do not always properly identify them as containing an Nvidia GPU.

If the default detection does not work, you can manually change the daemonset that the GPU operator creates by running the following command:

```
        nodeSelector:
          feature.node.kubernetes.io/pci-< class > _ < vendor>.present: "true"
```

where `class` is any 4 digit number starting with `03xy` and the vendor for Nvidia is `10de`. If this is already deployed, you can always change the `daemonset` and change the `nodeSelector` field so that it deploys to the right nodes.

899 https://github.com/NVIDIA/k8s-device-plugin/blob/master/README.md
900 https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html
901 https://nvidia.custhelp.com/app/answers/detail/a_id/131/kw/driver%20installation%20docs/related/1
902 https://kubernetes-sigs.github.io/node-feature-discovery/v0.7/get-started/features.html#pci

6.12.10.5.4.1 Next Topic

Update NVIDIA GPU Clusters (see page 1252)

6.12.10.5.5 Update NVIDIA GPU Clusters

Upgrading a node pool involves draining the existing nodes in the node pool and replacing them with new nodes. In order to ensure minimum downtime and maintain high availability of the critical application workloads during the upgrade process, we recommend deploying Pod Disruption Budget (Disruptions[903]) for your critical applications.

The Pod Disruption Budget will prevent any impact on critical applications as a result of misconfiguration or failures during the upgrade process.

6.12.10.5.5.1 Prerequisites:

- Deploy Pod Disruption Budget[904] (PDB)
- Konvoy Image Builder (see page 1626) (KIB)

6.12.10.5.5.2 Steps:

1. Deploy Pod Disruption Budget for your critical applications. If your application can tolerate only one replica to be unavailable at a time, then you can set Pod disruption budget as shown in the following example. The example below is for NVIDIA GPU node pools, but the process is the same for all node pools.

> Repeat this step for each additional node pool.

Create the file: `pod-disruption-budget-nvidia.yaml`

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: nvidia-critical-app
spec:
  maxUnavailable: 1
  selector:
    matchLabels:
      app: nvidia-critical-app
```

903 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/
904 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/

Apply the YAML file above using the following command:

```
kubectl create -f pod-disruption-budget-nvidia.yaml
```

2. Prepare OS image for your node pool using Konvoy Image Builder (see page 1626).

6.12.10.6 AWS Certificate Renewal

During cluster creation, Kubernetes establishes a Public Key Infrastructure (PKI) for generating the TLS certificates needed for securing cluster communication for various components such as `etcd`, `kubernetes-apiserver` and `kube-proxy`. The certificates created by these components have a default expiration of one year and are renewed when an administrator updates the cluster.

Kubernetes provides a facility to renew all certificates automatically during control plane updates. For administrators who need long-running clusters or clusters that are not upgraded often, `dkp` provides automated certificate renewal, without a cluster upgrade.

6.12.10.6.1 Prerequisites

- This feature requires Python 3.5 or greater to be installed on all control plane hosts.
- Complete the Bootstrap Cluster (see page 1187) topic.

6.12.10.6.2 Create a Cluster with Automated Certificate Renewal

To enable the automated certificate renewal, create a Konvoy cluster using the `certificate-renew-interval` flag:

```
dkp create cluster aws --certificate-renew-interval=60 --cluster-name=long-running
```

The `certificate-renew-interval` is the number of days after which Kubernetes-managed PKI certificates will be renewed. For example, an `certificate-renew-interval` value of 60 means the certificates will be renewed every 60 days.

6.12.10.6.3 Technical Details

The following manifests are modified on the control plane hosts, and are located at `/etc/kubernetes/manifests`. Modifications to these files requires SUDO access.

```
kube-controller-manager.yaml
kube-apiserver.yaml
```

```
kube-scheduler.yaml
kube-proxy.yaml
```

The following annotation indicates the time each component was reset:

```
metadata:
  annotations:
    konvoy.d2iq.io/restartedAt: $(date +%s)
```

This only occurs when the PKI certificates are older than the interval given at cluster creation time. This is activated by a `systemd` `timer` called `renew-certs.timer` that triggers an associated `systemd` service called `renew-certs.service` that runs on all of the control plane hosts.

6.12.10.6.4 Debug

To debug the automatic certificate renewal feature, a cluster administrator can look at several different components to see if the certificates were renewed. For example, an administrator might start with a look at the control plane pod definition to check the last reset time. To determine if a scheduler pod was properly reset, run the command:

```
kubectl get pod -n kube-system kube-scheduler-ip-10-0-xx-xx.us-west-2.compute.interna
l -o yaml
```

The output of the command will be similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    konvoy.d2iq.io/restartedAt: "1626124940.735733"
```

Administrators who want more details on the execution of the `systemd` service can use `ssh` to connect to the control plane hosts, and then use the `systemctl` and `journalctl` commands that follow to help diagnose potential issues.

To check the status of the timers, when they last ran, and when they are scheduled to run next, use the command:

```
systemctl list-timers
```

To check the status of the `renew-certs` service, use the command:

```
systemctl status renew-certs
```

To get the logs of the last run of the service, use the command:

```
journalctl logs -u renew-certs
```

6.12.10.6.5 Next Topic

Replace an AWS Node (see page 1255)

6.12.10.7 Replace an AWS Node

6.12.10.7.1 Prerequisites

Before you begin, you must:

- Create a workload cluster (see page 1190).
- Make the new cluster self-managed (see page 1227).

6.12.10.7.2 Replace a Worker Node

In certain situations, you may want to delete a worker node and have Cluster API[905] replace it with a newly provisioned machine.

1. Identify the name of the node to delete.

   List the nodes:

   ```
   kubectl --kubeconfig ${CLUSTER_NAME}.conf get nodes
   ```

   The output from this command resembles the following:

   ```
   NAME                                         STATUS   ROLES
   AGE   VERSION
   ip-10-0-100-85.us-west-2.compute.internal    Ready    <none>
   16m   v1.28.7
   ip-10-0-106-183.us-west-2.compute.internal   Ready    control-plane,master
   15m   v1.28.7
   ip-10-0-158-104.us-west-2.compute.internal   Ready    control-plane,master
   17m   v1.28.7
   ip-10-0-203-138.us-west-2.compute.internal   Ready    control-plane,master
   16m   v1.28.7
   ```

905 https://cluster-api.sigs.k8s.io/

```
ip-10-0-70-169.us-west-2.compute.internal    Ready    <none>
16m   v1.28.7
ip-10-0-77-176.us-west-2.compute.internal    Ready    <none>
16m   v1.28.7
ip-10-0-96-61.us-west-2.compute.internal     Ready    <none>
16m   v1.28.7
```

2. Export a variable with the node name to use in the next steps:

   This example uses the name `ip-10-0-100-85.us-west-2.compute.internal`.

   ```
   export NAME_NODE_TO_DELETE="<ip-10-0-100-85.us-west-2.compute.internal>"
   ```

3. Delete the Machine resource

   ```
   export NAME_MACHINE_TO_DELETE=$(kubectl --kubeconfig ${CLUSTER_NAME}.conf get
   machine -ojsonpath="{.items[?
   (@.status.nodeRef.name==\"$NAME_NODE_TO_DELETE\")].metadata.name}")
   kubectl --kubeconfig ${CLUSTER_NAME}.conf delete machine
   "$NAME_MACHINE_TO_DELETE"
   ```

   ```
   machine.cluster.x-k8s.io "aws-example-1-md-0-cb9c9bbf7-t894m" deleted
   ```

   The command will not return immediately. It will return after the Machine resource has been deleted.

   A few minutes after the Machine resource is deleted, the corresponding Node resource is also deleted.

4. Observe that the Machine resource is being replaced using this command:

   ```
   kubectl --kubeconfig ${CLUSTER_NAME}.conf get machinedeployment
   ```

   ```
   NAME               CLUSTER       REPLICAS   READY   UPDATED   UNAVAILABLE
   PHASE       AGE     VERSION
   aws-example-md-0   aws-example   4          3       4         1
   ScalingUp   7m53s   v1.26.3
   ```

In this example, there are two replicas, but only 1 is ready. One replica is unavailable, and the `ScalingUp` phase means a new Machine is being created.

5. Identify the replacement Machine using this command:

```
export NAME_NEW_MACHINE=$(kubectl --kubeconfig ${CLUSTER_NAME}.conf get
machines \
    -l=cluster.x-k8s.io/deployment-name=${CLUSTER_NAME}-md-0 \
    -ojsonpath='{.items[?(@.status.phase=="Running")].metadata.name}{"\n"}')
echo "$NAME_NEW_MACHINE"
```

```
aws-example-md-0-cb9c9bbf7-hcl8z aws-example-md-0-cb9c9bbf7-rtdqw aws-example-
md-0-cb9c9bbf7-td29r aws-example-md-0-cb9c9bbf7-w64kg
```

If the output is empty, the new Machine has probably exited the `Provisioning` phase and entered the `Running` phase.

6. Identify the replacement Node using this command:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf get nodes
```

```
NAME                                         STATUS   ROLES
AGE   VERSION
ip-10-0-106-183.us-west-2.compute.internal   Ready    control-plane,master
20m   v1.28.7
ip-10-0-158-104.us-west-2.compute.internal   Ready    control-plane,master
23m   v1.28.7
ip-10-0-203-138.us-west-2.compute.internal   Ready    control-plane,master
22m   v1.28.7
ip-10-0-70-169.us-west-2.compute.internal    Ready    <none>
22m   v1.28.7
ip-10-0-77-176.us-west-2.compute.internal    Ready    <none>
22m   v1.28.7
ip-10-0-86-58.us-west-2.compute.internal     Ready    <none>
57s   v1.28.7
ip-10-0-96-61.us-west-2.compute.internal     Ready    <none>
22m   v1.28.7
```

If the output is empty, the Node resource is not yet available, or does not yet have the expected annotation. Wait a few minutes, then repeat the command.

6.12.11 Configure Infrastructure in UI

This page refers to working inside the AWS Console in order to add Infrastructure Provider.

6.12.11.1 Create Infrastructure Provider

To configure an AWS Provider with a User Role in the AWS Console (UI), perform the following steps:

1. From the top menu bar, select your target workspace.
2. Select **Infrastructure Providers** in the **Administration** section of the sidebar menu.
3. Select the **Add Infrastructure Provider** button.
4. Select the **Amazon Web Services (AWS)** option.
5. Ensure **Role** is selected as the **Authentication Method**.
6. Enter a name for your infrastructure provider. Select a name that matches the AWS user.
7. Enter the **Role ARN**.
8. You can add an **External ID** if you share the Role with a 3rd party. External IDs secure your environment from accidentally used roles. Read more about External IDs[906].
9. Select **Save** to save your provider.

6.12.11.2 Fill out the Add Infrastructure Provider Form

To fill out the Add Infrastructure Provider form in the AWS Console using Static Credentials, perform the following steps:

1. In Kommander, select the Workspace associated with the credentials you are adding.
2. Navigate to **Administration > Infrastructure Providers** and click the **Add Infrastructure Provider** button.
3. Select the Amazon Web Services (AWS) option.
4. Ensure **Static** is selected as the Authentication Method.
5. Enter a name for your infrastructure provider for later reference. Consider choosing a name that matches the AWS user.
6. Fill out the access and secret keys using the keys generated above.
7. Select **Save** to save your provider.

906 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-user_externalid.html

6.13 EKS Infrastructure

Enterprise Gov Advanced

When installing DKP on your EKS infrastructure, you must set up various permissions also through AWS Infrastructure (see page 1148).

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

The steps for creating and accessing your cluster are listed below:

6.13.1 EKS Introduction

Enterprise Gov Advanced

DKP brings value to EKS customers by providing all components needed for a production-ready Kubernetes environment. DKP provides the capability to provision EKS clusters using the DKP UI, In addition to above. It also provides the ability to upgrade your EKS clusters using the DKP platform, making it possible to manage the complete lifecycle of EKS clusters from a centralized platform.

DKP adds value to Amazon EKS through:

- **Time to Value** in hours/days to get to production, instead of weeks/months, or even failure. Particularly in complex environments like air-gapped, customers tried various options and even after spending millions did not see success, or saw Day 2 later than they could have. We delivered results in hours/days.
- **Less Risk**
    - **Cloud-Native Expertise** eliminates the lack of skills issue. Our industry-leading expertise closes skill gaps on the customer side, avoids costly mistakes, transfers skills, and improves project success rates while shortening timelines.
    - **Simplicity** mitigates operational complexity. We focus on a great user experience and automate the hard parts of cloud-native operations to get customers to Day 2 faster and meet

all Day 2 operational challenges. This frees up time on the customer side to build what differentiates them, instead of reinventing the wheel for Kubernetes operations.

    - **Military-Grade Security** alleviates security concerns. The D2iQ Kubernetes Platform can be configured to meet NSA Kubernetes security hardening guidelines. D2iQ Kubernetes Platform and supported add-on components are security scanned and secure out of the box. Encryption of data-at-rest, FIPS compliance, and fully supported air-gapped deployments round out D2iQ offerings.
- **Lower TCO** with operational insights and a simpler platform that curates needed capabilities from Amazon EKS and the open source community that reduces the time and cost of consulting engagements as well as ongoing support costs.
- **Enterprise-grade Kubernetes** - comes with a curated list of Day 2 applications necessary for running Kubernetes in production.
- **One platform for all** - Single platform to manage multiple clusters on any infrastructure cloud, on-premise, and edge.
- **D2IQ GitOps and EKS** - Delivering business value through applications is the primary goal of any Kubernetes cluster. While EKS provides the hosted framework that leads the market, delivering applications to your environment requires a mature and integrated approach. D2IQ DKP provides workspace and project level constructs to a Kubernetes cluster so that application teams have division of resources, security and cost optimization at the the project and namespace level.
    - Projects deliver applications via the built in GitOps via FluxCD - Just provide a Git repository and DKP does the rest
    - Through integration with Kubecost, DKP monitors utilization of project resources and proves real time reporting for performance and cost optimization
    - Security of projects is defined through DKP integration of customer authentication methods and is reinforced through several layers of application security
- **Cluster Lifecycle Management through CAPI -** Through the use of cluster API, DKP gives customers full lifecycle management of their EKS clusters with the ability to instantiate new EKS clusters through a unified API. This allows administrators to deploy new EKS clusters through code and deliver consistent cluster configurations.
- Time to application value is greatly reduced by minimizing the steps necessary to provision a cluster segment clusters through integrated permissions
- Secure and reliable cluster deployments
- Automatic day 2 operations of EKS clusters (Monitoring, Logging, Central Management, Security, Cost Optimization)
- Day 2 GitOps integration with every EKS cluster

6.13.1.1 Next Step

EKS Prerequisites and Permissions (see page 1261)

6.13.2 EKS Prerequisites and Permissions

6.13.2.1 Konvoy Prerequisites

Before you begin using Konvoy, you must have:

- An x86_64-based Linux or macOS machine.
- The `dkp` binary for Linux, or macOS.
- A Container engine/runtime installed is required to install DKP:
    - Version Docker®[907] container engine version 18.09.2 or higher installed for Linux or MacOS - On macOS, Docker runs in a virtual machine which needs configured with at least 8 GB of memory.
    - Version 4.0 of Podman[908] or higher for Linux. Host requirements found here: Host Requirements[909]
- kubectl[910] for interacting with the running cluster.
- A valid AWS account with credentials configured[911].
- For a local registry whether air-gapped or non-air-gapped environment, download and extract the bundle. Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.7.0_linux_amd64.tar.gz`) to load registry.

On macOS, Docker runs in a virtual machine. Configure this virtual machine with at least 8GB of memory.

6.13.2.2 Control Plane Nodes

You should have at least three control plane nodes. Each control plane node should have at least:

- 4 cores
- 16 GiB memory
- Approximately 80 GiB of free space for the volume used for /var/lib/kubelet and /var/lib/containerd.
- Disk usage must be below 85% on the root volume.

DKP on AWS defaults to deploying an `m5.xlarge` instance with an 80GiB root volume for control plane nodes, which meets the above requirements.

907 https://docs.docker.com/get-docker/
908 https://podman.io/getting-started/installation
909 https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
910 https://kubernetes.io/docs/tasks/tools/#kubectl
911 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html

6.13.2.3 Worker Nodes

You should have at least four worker nodes. The specific number of worker nodes required for your environment can vary depending on the cluster workload and size of the nodes. Each worker node should have at least:

- 8 cores
- 32 GiB memory
- Around 80 GiB of free space for the volume used for /var/lib/kubelet and /var/lib/containerd.
- Disk usage must be below 85% on the root volume.

DKP on AWS defaults to deploying a `m5.2xlarge` instance with an 80GiB root volume for worker nodes, which meets the above requirements.

If you use these instructions to create a cluster on AWS using the DKP default settings without any edits to configuration files or additional flags, your cluster is deployed on an Ubuntu 20.04 operating system image (see page 67) with 3 control plane nodes, and 4 worker nodes which match the requirements above.

6.13.2.4 AWS Prerequisites

Before you begin using Konvoy with AWS, you must:

- Create an IAM policy configuration (see page 1156).
- Create the Cluster IAM Policies (see page 1162) in your AWS account.
- The user you delegate from your role must have a minimum set of permissions, see AWS Minimal Permissions and Role to Create Clusters (see page 1156) page for AWS.
- Export the AWS region where you want to deploy the cluster:

```
export AWS_REGION=us-west-2
```

- Export the AWS profile with the credentials you want to use to create the Kubernetes cluster:

```
export AWS_PROFILE=<profile>
```

6.13.2.5 Related Topics:

AWS Konvoy Image Builder (see page 1153)

6.13.2.6 Next Step:

Minimal User Permission for EKS Cluster Creation (see page 1263)

6.13.2.7 Minimal User Permission for EKS Cluster Creation

The following is a CloudFormation stack which adds a policy named `eks-bootstrapper` to manage EKS cluster to the `dkp-bootstrapper-role` created by the CloudFormation stack for AWS in the Minimal Permissions and Role to Create Cluster (see page 1156) section.

Consult the Leveraging the Role (see page 1160)section for an example of how to use this role and how a system administrator wants to expose using the permissions.

6.13.2.7.1 EKS CloudFormation stack:

```
AWSTemplateFormatVersion: 2010-09-09
Parameters:
  existingBootstrapperRole:
    Type: CommaDelimitedList
    Description: 'Name of existing minimal role you want to add to add EKS cluster
management permissions to'
    Default: dkp-bootstrapper-role
Resources:
  EKSMinimumPermissions:
    Properties:
      Description: Minimal user policy to manage eks clusters
      ManagedPolicyName: eks-bootstrapper
      PolicyDocument:
        Statement:
          - Action:
              - 'ssm:GetParameter'
            Effect: Allow
            Resource:
              - 'arn:*:ssm:*:*:parameter/aws/service/eks/optimized-ami/*'
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:*:iam::*:role/aws-service-role/eks.amazonaws.com/
AWSServiceRoleForAmazonEKS
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks-nodegroup.amazonaws.com
            Effect: Allow
            Resource:
              - >-
```

```
                arn:*:iam::*:role/aws-service-role/eks-nodegroup.amazonaws.com/
AWSServiceRoleForAmazonEKSNodegroup
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks-fargate.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:aws:iam::*:role/aws-service-role/eks-fargate-pods.amazonaws.com/
AWSServiceRoleForAmazonEKSForFargate
          - Action:
              - 'iam:GetRole'
              - 'iam:ListAttachedRolePolicies'
            Effect: Allow
            Resource:
              - 'arn:*:iam::*:role/*'
          - Action:
              - 'iam:GetPolicy'
            Effect: Allow
            Resource:
              - 'arn:aws:iam::aws:policy/AmazonEKSClusterPolicy'
          - Action:
              - 'eks:DescribeCluster'
              - 'eks:ListClusters'
              - 'eks:CreateCluster'
              - 'eks:TagResource'
              - 'eks:UpdateClusterVersion'
              - 'eks:DeleteCluster'
              - 'eks:UpdateClusterConfig'
              - 'eks:UntagResource'
              - 'eks:UpdateNodegroupVersion'
              - 'eks:DescribeNodegroup'
              - 'eks:DeleteNodegroup'
              - 'eks:UpdateNodegroupConfig'
              - 'eks:CreateNodegroup'
              - 'eks:AssociateEncryptionConfig'
              - 'eks:ListIdentityProviderConfigs'
              - 'eks:AssociateIdentityProviderConfig'
              - 'eks:DescribeIdentityProviderConfig'
              - 'eks:DisassociateIdentityProviderConfig'
            Effect: Allow
            Resource:
              - 'arn:*:eks:*:*:cluster/*'
              - 'arn:*:eks:*:*:nodegroup/*/*/*'
          - Action:
              - 'ec2:AssociateVpcCidrBlock'
              - 'ec2:DisassociateVpcCidrBlock'
              - 'eks:ListAddons'
              - 'eks:CreateAddon'
              - 'eks:DescribeAddonVersions'
```

```
            - 'eks:DescribeAddon'
            - 'eks:DeleteAddon'
            - 'eks:UpdateAddon'
            - 'eks:TagResource'
            - 'eks:DescribeFargateProfile'
            - 'eks:CreateFargateProfile'
            - 'eks:DeleteFargateProfile'
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - 'iam:PassRole'
          Condition:
            StringEquals:
              'iam:PassedToService': eks.amazonaws.com
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - 'kms:CreateGrant'
            - 'kms:DescribeKey'
          Condition:
            'ForAnyValue:StringLike':
              'kms:ResourceAliases': alias/cluster-api-provider-aws-*
          Effect: Allow
          Resource:
            - '*'
      Version: 2012-10-17
    Roles: !Ref existingBootstrapperRole
  Type: 'AWS::IAM::ManagedPolicy'
```

If your role is not named `dkp-bootstrapper-role` change the parameter on line 6 of the file.

To create the resources in the cloudformation stack, copy the contents above into a file. Before executing the following command, replace `MYFILENAME.yaml` and `MYSTACKNAME` with the intended values for your system:

```
aws cloudformation create-stack --template-body=file://MYFILENAME.yaml --stack-
name=MYSTACKNAME --capabilities CAPABILITY_NAMED_IAM
```

6.13.2.7.2 Next Step:

EKS Cluster IAM Permissions and Roles (see page 1266)

6.13.2.8 EKS Cluster IAM Permissions and Roles

This section guides a DKP user in creating IAM Policies and Instance Profiles that governs who has access to the cluster. The IAM Role is used by the cluster's control plane and worker nodes using the provided AWS CloudFormation Stack specific to EKS. This CloudFormation Stack has additional permissions that are used to delegate access roles for other users.

6.13.2.8.1 Prerequisites from AWS:

- The user you delegate from your role must have a minimum set of permissions, see User Roles and Instance Profiles (see page 1156) page for AWS.
- Create the Cluster IAM Policies (see page 1162) in your AWS account.

6.13.2.8.1.1 EKS IAM Artifacts

Policies

- `controllers-eks.cluster-api-provider-aws.sigs.k8s.io` - enumerates the Actions required by the workload cluster to create and modify EKS clusters in the user's AWS Account. It is attached to the existing `control-plane.cluster-api-provider-aws.sigs.k8s.io` role
- `eks-nodes.cluster-api-provider-aws.sigs.k8s.io` - enumerates the Actions required by the EKS workload cluster's worker machines. It is attached to the existing `nodes.cluster-api-provider-aws.sigs.k8s.io`

Roles

- `eks-controlplane.cluster-api-provider-aws.sigs.k8s.io` - is the Role associated with EKS cluster control planes

> NOTE: `control-plane.cluster-api-provider-aws.sigs.k8s.io` and `nodes.cluster-api-provider-aws.sigs.k8s.io` roles were created by Cluster IAM Policies and Roles (see page 1162) in AWS.

Below is a CloudFormation stack[912] that includes IAM policies and roles required to setup EKS Clusters:

```
AWSTemplateFormatVersion: 2010-09-09
```

912 https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html

```
Parameters:
  existingControlPlaneRole:
    Type: CommaDelimitedList
    Description: 'Names of existing Control Plane Role you want to add to the newly
created EKS Managed Policy for AWS cluster API controllers'
    Default: control-plane.cluster-api-provider-aws.sigs.k8s.io
  existingNodeRole:
    Type: CommaDelimitedList
    Description: 'ARN of the Nodes Managed Policy to add to the role for nodes'
    Default: nodes.cluster-api-provider-aws.sigs.k8s.io
Resources:
  AWSIAMManagedPolicyControllersEKS:
    Properties:
      Description: For the Kubernetes Cluster API Provider AWS Controllers
      ManagedPolicyName: controllers-eks.cluster-api-provider-aws.sigs.k8s.io
      PolicyDocument:
        Statement:
          - Action:
              - 'ssm:GetParameter'
            Effect: Allow
            Resource:
              - 'arn:*:ssm:*:*:parameter/aws/service/eks/optimized-ami/*'
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:*:iam::*:role/aws-service-role/eks.amazonaws.com/
AWSServiceRoleForAmazonEKS
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks-nodegroup.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:*:iam::*:role/aws-service-role/eks-nodegroup.amazonaws.com/
AWSServiceRoleForAmazonEKSNodegroup
          - Action:
              - 'iam:CreateServiceLinkedRole'
            Condition:
              StringLike:
                'iam:AWSServiceName': eks-fargate.amazonaws.com
            Effect: Allow
            Resource:
              - >-
                arn:aws:iam::*:role/aws-service-role/eks-fargate-pods.amazonaws.com/
AWSServiceRoleForAmazonEKSForFargate
```

```
- Action:
    - 'iam:GetRole'
    - 'iam:ListAttachedRolePolicies'
  Effect: Allow
  Resource:
    - 'arn:*:iam::*:role/*'
- Action:
    - 'iam:GetPolicy'
  Effect: Allow
  Resource:
    - 'arn:aws:iam::aws:policy/AmazonEKSClusterPolicy'
- Action:
    - 'eks:DescribeCluster'
    - 'eks:ListClusters'
    - 'eks:CreateCluster'
    - 'eks:TagResource'
    - 'eks:UpdateClusterVersion'
    - 'eks:DeleteCluster'
    - 'eks:UpdateClusterConfig'
    - 'eks:UntagResource'
    - 'eks:UpdateNodegroupVersion'
    - 'eks:DescribeNodegroup'
    - 'eks:DeleteNodegroup'
    - 'eks:UpdateNodegroupConfig'
    - 'eks:CreateNodegroup'
    - 'eks:AssociateEncryptionConfig'
    - 'eks:ListIdentityProviderConfigs'
    - 'eks:AssociateIdentityProviderConfig'
    - 'eks:DescribeIdentityProviderConfig'
    - 'eks:DisassociateIdentityProviderConfig'
  Effect: Allow
  Resource:
    - 'arn:*:eks:*:*:cluster/*'
    - 'arn:*:eks:*:*:nodegroup/*/*/*'
- Action:
    - 'ec2:AssociateVpcCidrBlock'
    - 'ec2:DisassociateVpcCidrBlock'
    - 'eks:ListAddons'
    - 'eks:CreateAddon'
    - 'eks:DescribeAddonVersions'
    - 'eks:DescribeAddon'
    - 'eks:DeleteAddon'
    - 'eks:UpdateAddon'
    - 'eks:TagResource'
    - 'eks:DescribeFargateProfile'
    - 'eks:CreateFargateProfile'
    - 'eks:DeleteFargateProfile'
  Effect: Allow
  Resource:
    - '*'
- Action:
    - 'iam:PassRole'
```

```
          Condition:
            StringEquals:
              'iam:PassedToService': eks.amazonaws.com
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - 'kms:CreateGrant'
            - 'kms:DescribeKey'
          Condition:
            'ForAnyValue:StringLike':
              'kms:ResourceAliases': alias/cluster-api-provider-aws-*
          Effect: Allow
          Resource:
            - '*'
      Version: 2012-10-17
    Roles: !Ref existingControlPlaneRole
  Type: 'AWS::IAM::ManagedPolicy'
AWSIAMManagedEKSNodesPolicy:
  Properties:
    Description: Additional Policies to nodes role to work for EKS
    ManagedPolicyName: eks-nodes.cluster-api-provider-aws.sigs.k8s.io
    PolicyDocument:
      Statement:
        - Action:
            - "ec2:AssignPrivateIpAddresses"
            - "ec2:AttachNetworkInterface"
            - "ec2:CreateNetworkInterface"
            - "ec2:DeleteNetworkInterface"
            - "ec2:DescribeInstances"
            - "ec2:DescribeTags"
            - "ec2:DescribeNetworkInterfaces"
            - "ec2:DescribeInstanceTypes"
            - "ec2:DetachNetworkInterface"
            - "ec2:ModifyNetworkInterfaceAttribute"
            - "ec2:UnassignPrivateIpAddresses"
          Effect: Allow
          Resource:
            - '*'
        - Action:
            - ec2:CreateTags
          Effect: Allow
          Resource:
            - arn:aws:ec2:*:*:network-interface/*
        - Action:
            - "ec2:DescribeInstances"
            - "ec2:DescribeInstanceTypes"
            - "ec2:DescribeRouteTables"
            - "ec2:DescribeSecurityGroups"
            - "ec2:DescribeSubnets"
            - "ec2:DescribeVolumes"
            - "ec2:DescribeVolumesModifications"
```

```
              - "ec2:DescribeVpcs"
              - "eks:DescribeCluster"
            Effect: Allow
            Resource:
              - '*'
        Version: 2012-10-17
      Roles: !Ref existingNodeRole
    Type: 'AWS::IAM::ManagedPolicy'
  AWSIAMRoleEKSControlPlane:
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action:
              - 'sts:AssumeRole'
            Effect: Allow
            Principal:
              Service:
                - eks.amazonaws.com
        Version: 2012-10-17
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/AmazonEKSClusterPolicy'
      RoleName: eks-controlplane.cluster-api-provider-aws.sigs.k8s.io
    Type: 'AWS::IAM::Role'
```

To create the resources in the CloudFormation stack, copy the contents above into a file and execute the following command after replacing `MYFILENAME.yaml` and `MYSTACKNAME` with the intended values:

```
 aws cloudformation create-stack --template-body=file://MYFILENAME.yaml --stack-
name=MYSTACKNAME --capabilities  CAPABILITY_NAMED_IAM
```

Add EKS CSI Policy

AWS CloudFormation does not support attaching an existing IAM Policy to an existing IAM Role. Add the necessary IAM policy to your worker instance profile using the `aws` CLI:

```
aws iam attach-role-policy --role-name nodes.cluster-api-provider-aws.sigs.k8s.io --
policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy
```

6.13.2.8.1.2 Next Step

Create an EKS Cluster from the CLI (see page 1271)

6.13.3 Create an EKS Cluster from the CLI

Enterprise Gov Advanced

> Ensure that the `KUBECONFIG` environment variable is set to the Management cluster by running `export KUBECONFIG=<Management_cluster_kubeconfig>.conf`.

6.13.3.1 Create a New EKS Kubernetes Cluster from the CLI

If you prefer to work in the shell, you can continue by creating a new cluster following these steps. If you prefer to log in to the DKP UI, you can create a new cluster from there using the steps on this page: Create an EKS Cluster from the DKP UI (see page 1278)

> By default, the control-plane Nodes will be created in 3 different Availability Zones. However, the default worker Nodes will reside in a single zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command.

Follow these steps:

1. Set the environment variable to the name you assigned this cluster.

   ```
   export CLUSTER_NAME=eks-example
   ```

2. Make sure your AWS credentials are up-to-date. Refresh the credentials command is only necessary if you are using Static Credentials (see page 0) (Access Keys) otherwise, if you are using role-based authentication on a bastion host, proceed to step 3:

   ```
   dkp update bootstrap credentials aws
   ```

3. Create the cluster:

   ```
   dkp create cluster eks \
     --cluster-name=${CLUSTER_NAME} \
     --additional-tags=owner=$(whoami)
   ```

⚠ **NOTE**: If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

⚠ **Optional flag for ECR**: Configure your cluster to use an existing local registry as a mirror when attempting to pull images. Below is an AWS ECR example:

```
export --registry-mirror-url-string=<YOUR_ECR_URL>
```

- `REGISTRY-MIRROR-URL-STRING`: the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images. Users can still pull their own images from ECR directly or use ECR as a mirror. See CLI commands for `--registry-mirror` flag here: dkp create cluster eks (see page 1854)

```
Generating cluster resources
cluster.cluster.x-k8s.io/eks-example created
awsmanagedcontrolplane.controlplane.cluster.x-k8s.io/eks-example-control-plane
 created
machinedeployment.cluster.x-k8s.io/eks-example-md-0 created
awsmachinetemplate.infrastructure.cluster.x-k8s.io/eks-example-md-0 created
eksconfigtemplate.bootstrap.cluster.x-k8s.io/eks-example-md-0 created
clusterresourceset.addons.cluster.x-k8s.io/calico-cni-installation-eks-example
 created
configmap/calico-cni-installation-eks-example created
configmap/tigera-operator-eks-example created
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-eks-example
 created
configmap/cluster-autoscaler-eks-example created
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-eks-example
 created
configmap/node-feature-discovery-eks-example created
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-eks-example
 created
configmap/nvidia-feature-discovery-eks-example created
```

4. Inspect or edit the cluster objects:

   Use your favorite editor.

   NOTE: Editing the cluster objects requires some understanding of Cluster API. Edits can prevent the cluster from deploying successfully.

   The objects are Custom Resources[913] defined by Cluster API components, and they belong in three different categories:

   a. **Cluster**
   A *Cluster* object has references to the infrastructure-specific and control plane objects. Because this is an AWS cluster, there is an *AWSCluster* object that describes the

913 https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/

infrastructure-specific cluster properties. Here, this means the AWS region, the VPC ID, subnet IDs, and security group rules required by the Pod network implementation.

b. **Control Plane**
A *AWSManagedControlPlane* object describes the control plane, which is the group of machines that run the Kubernetes control plane components, which include the etcd distributed database, the API server, the core controllers, and the scheduler. The object describes the configuration for these components. The object also has a reference to an infrastructure-specific object that describes the properties of all control plane machines.

c. **Node Pool**
A Node Pool is a collection of machines with identical properties. For example, a cluster might have one Node Pool with large memory capacity, another Node Pool with GPU support. Each Node Pool is described by three objects: The MachinePool references an object that describes the configuration of Kubernetes components (for example, kubelet) deployed on each node pool machine, and an infrastructure-specific object that describes the properties of all node pool machines. Here, it references a *KubeadmConfigTemplate*, and an *AWSMachineTemplate* object, which describes the instance type, the type of disk used, the size of the disk, among other properties.

For in-depth documentation about the objects, read Concepts[914] in the Cluster API Book.

5. Wait for the cluster control-plane to be ready:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=20m
```

```
cluster.cluster.x-k8s.io/eks-example condition met
```

The `READY` status will become `True` after the cluster control-plane becomes ready in one of the following steps.

6. After the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. Konvoy provides a command to describe the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

```
NAME                                                           READY
SEVERITY  REASON  SINCE  MESSAGE
```

914 https://cluster-api.sigs.k8s.io/user/concepts.html

```
Cluster/eks-example                                                    True
10m
├─ControlPlane - AWSManagedControlPlane/eks-example-control-plane  True
10m
└─Workers
  └─MachineDeployment/eks-example-md-0                                 True
26s
    ├─Machine/eks-example-md-0-78fcd7c7b7-66ntt                        True
84s
    ├─Machine/eks-example-md-0-78fcd7c7b7-b9qmc                        True
84s
    ├─Machine/eks-example-md-0-78fcd7c7b7-v5vfq                        True
84s
    └─Machine/eks-example-md-0-78fcd7c7b7-zl6m2                        True
84s
```

7. As they progress, the controllers also create Events. List the Events using this command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, the example uses `grep`. It is also possible to use separate commands to get Events for specific objects. For example, `kubectl get events --field-selector involvedObject.kind="AWSCluster"` and `kubectl get events --field-selector involvedObject.kind="AWSMachine"`.

```
46m         Normal   SuccessfulCreateVPC
awsmanagedcontrolplane/eks-example-control-plane   Created new managed VPC
"vpc-05e775702092abf09"
46m         Normal   SuccessfulSetVPCAttributes
awsmanagedcontrolplane/eks-example-control-plane   Set managed VPC attributes
for "vpc-05e775702092abf09"
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0419dd3f2dfd95ff8"
46m         Normal   SuccessfulModifySubnetAttributes
awsmanagedcontrolplane/eks-example-control-plane   Modified managed Subnet
"subnet-0419dd3f2dfd95ff8" attributes
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0e724b128e3113e47"
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-06b2b31ea6a8d3962"
46m         Normal   SuccessfulModifySubnetAttributes
awsmanagedcontrolplane/eks-example-control-plane   Modified managed Subnet
"subnet-06b2b31ea6a8d3962" attributes
```

```
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0626ce238be32bf98"
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0f53cf59f83177800"
46m         Normal   SuccessfulModifySubnetAttributes
awsmanagedcontrolplane/eks-example-control-plane   Modified managed Subnet
"subnet-0f53cf59f83177800" attributes
46m         Normal   SuccessfulCreateSubnet
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Subnet
"subnet-0878478f6bbf153b2"
46m         Normal   SuccessfulCreateInternetGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new managed Internet
Gateway "igw-09fb52653949d4579"
46m         Normal   SuccessfulAttachInternetGateway
awsmanagedcontrolplane/eks-example-control-plane   Internet Gateway
"igw-09fb52653949d4579" attached to VPC "vpc-05e775702092abf09"
46m         Normal   SuccessfulCreateNATGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new NAT Gateway
"nat-06356aac28079952d"
46m         Normal   SuccessfulCreateNATGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new NAT Gateway
"nat-0429d1cd9d956bf35"
46m         Normal   SuccessfulCreateNATGateway
awsmanagedcontrolplane/eks-example-control-plane   Created new NAT Gateway
"nat-059246bcc9d4e88e7"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-01689c719c484fd3c"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-01689c719c484fd3c" with subnet "subnet-0419dd3f2dfd95ff8"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-065af81b9752eeb69"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-065af81b9752eeb69" with subnet "subnet-0e724b128e3113e47"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-03eeff810a89afc98"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-03eeff810a89afc98" with subnet "subnet-06b2b31ea6a8d3962"
```

```
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-0fab36f8751fdee73"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-0fab36f8751fdee73" with subnet "subnet-0626ce238be32bf98"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-0e5c9c7bbc3740a0f"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-0e5c9c7bbc3740a0f" with subnet "subnet-0f53cf59f83177800"
46m         Normal   SuccessfulCreateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Created managed RouteTable
"rtb-0bf58eb5f73c387af"
46m         Normal   SuccessfulCreateRoute
awsmanagedcontrolplane/eks-example-control-plane   Created route {...
46m         Normal   SuccessfulAssociateRouteTable
awsmanagedcontrolplane/eks-example-control-plane   Associated managed
RouteTable "rtb-0bf58eb5f73c387af" with subnet "subnet-0878478f6bbf153b2"
46m         Normal   SuccessfulCreateSecurityGroup
awsmanagedcontrolplane/eks-example-control-plane   Created managed
SecurityGroup "sg-0b045c998a120a1b2" for Role "node-eks-additional"
46m         Normal   InitiatedCreateEKSControlPlane
awsmanagedcontrolplane/eks-example-control-plane   Initiated creation of a new
 EKS control plane default_eks-example-control-plane
37m         Normal   SuccessfulCreateEKSControlPlane
awsmanagedcontrolplane/eks-example-control-plane   Created new EKS control
plane default_eks-example-control-plane
37m         Normal   SucessfulCreateKubeconfig
awsmanagedcontrolplane/eks-example-control-plane   Created kubeconfig for
 cluster "eks-example"
37m         Normal   SucessfulCreateUserKubeconfig
awsmanagedcontrolplane/eks-example-control-plane   Created user kubeconfig for
 cluster "eks-example"
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-4t9nc                  Created new node instance
with id "i-0aecc1897c93df740"
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-4t9nc                  AWS Secret entries
containing userdata deleted
26m         Normal   SuccessfulSetNodeRef                            machine/
eks-example-md-0-78fcd7c7b7-fn7x9          ip-10-0-88-24.us-west-2.compute.inte
rnal
26m         Normal   SuccessfulSetNodeRef                            machine/
eks-example-md-0-78fcd7c7b7-g64nv          ip-10-0-110-219.us-west-2.compute.in
ternal
```

```
26m         Normal   SuccessfulSetNodeRef                machine/
eks-example-md-0-78fcd7c7b7-gwc5j          ip-10-0-101-161.us-west-2.compute.in
ternal
26m         Normal   SuccessfulSetNodeRef                machine/
eks-example-md-0-78fcd7c7b7-j58s4          ip-10-0-127-49.us-west-2.compute.int
ernal
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-fn7x9"
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-g64nv"
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-j58s4"
46m         Normal   SuccessfulCreate
machineset/eks-example-md-0-78fcd7c7b7             Created machine "eks-
example-md-0-78fcd7c7b7-gwc5j"
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-7whkv                  Created new node instance
with id "i-06dfc0466b8f26695"
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-7whkv                  AWS Secret entries
containing userdata deleted
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-ttgzv                  Created new node instance
with id "i-0544fce0350fd41fb"
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-ttgzv                  AWS Secret entries
containing userdata deleted
27m         Normal   SuccessfulCreate
awsmachine/eks-example-md-0-v2hrf                  Created new node instance
with id "i-0498906edde162e59"
26m         Normal   SuccessfulDeleteEncryptedBootstrapDataSecrets
awsmachine/eks-example-md-0-v2hrf                  AWS Secret entries
containing userdata deleted
46m         Normal   SuccessfulCreate
machinedeployment/eks-example-md-0                 Created MachineSet "eks-
example-md-0-78fcd7c7b7"
```

6.13.3.2 Known Limitations

Be aware of these limitations in the current release of Konvoy.

- The Konvoy version used to create a workload cluster must match the Konvoy version used to delete a workload cluster.

- EKS clusters cannot be Self-managed.
- Konvoy supports deploying one workload cluster.
- Konvoy generates a set of objects for one Node Pool.
- Konvoy does not validate edits to cluster objects.

6.13.3.2.1 Next Step

Grant Cluster Access (see page 1280)

6.13.3.3 Create an EKS Cluster from the DKP UI

Enterprise Gov Advanced

The DKP user interface allows you to provision a Cluster from your browser quickly and easily.

6.13.3.3.1 Create an AWS Infrastructure Provider

Before you create a Cluster, you first need to create an AWS infrastructure provider to hold your AWS/EKS Credentials:

1. Get the AWS RoleARN (see page 1156).

   ```
   aws iam get-role --role-name <role-name> --query 'Role.[RoleName, Arn]' --
   output text
   ```

2. Select **Infrastructure Providers** from the Dashboard menu.
3. Select **Add Infrastructure Provider**.
4. Choose a workspace. If you are already in a workspace, the provider is automatically created in that workspace.
5. Ensure you select **Amazon Web Services**.
6. Add a **Name** for your Infrastructure Provider and include the **Role ARN** from Step 1 above.
7. Select **Save**.

> ⚠ If you choose to, you can use static credentials. However, this method is not as secure so it is not recommended.

6.13.3.3.2 Provision an EKS Cluster

Follow these steps to provision the EKS cluster:

1. From the top menu bar, select your target workspace.
2. Select **Clusters > Add Cluster**.
   This begins the provisioning workflow.
3. Choose **Create Cluster**.
4. Enter the **Cluster Name**.
5. Select **EKS** from the **Choose Infrastructure** choices.
6. If available, choose a **Kubernetes Version**. Otherwise, the default Kubernetes version[915] installs.
7. Select a data center **region** or specify a custom **region**.
8. Edit your worker **Node Pools** as necessary. You can choose the **Number of Nodes**, the **Machine Type**, and our **IAM Instance Profile**. For the worker pool, you can also choose a **Worker Availability Zone**.
9. Add any additional **Labels** or **Infrastructure Provider Tags** as necessary.
10. Validate your inputs, and then select **Create**.

You are redirected to the **Clusters** page, where you see your **Cluster** in the **Provisioning** status. Hover over the status to view the details.

After about 15 minutes, your **Cluster** should be in the **Provisioned** status.

See AWS RoleARN[916] for more information from the AWS site.

6.13.3.3.3 Access EKS Cluster

After the cluster is successfully attached(managed), you can retrieve a custom `kubeconfig` file from the UI using your Kommander administrator credentials.

6.13.3.3.4 IAM User and Role Access for EKS Clusters

When creating an EKS cluster through the UI, the `kubeconfig` that is returned using the download `kubeconfig` button allows access for 15 minutes. To follow best practices for AWS security, you should configure accessing the EKS cluster using IAM role or user based authentication. This allows account administrators to monitor all actions made.

To enable IAM based cluster access, follow the steps below:

1. Download the `kubeconfig` by selecting the Download `kubeconfig` button on the top section of the UI.
2. Using that `kubeconfig`, edit the config map with a command similar to the one below:

```
kubectl --kubeconfig=MYCLUSTER.conf edit cm -n kube-system aws-auth
```

915 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/150765792/DKP+2.5.0+Supported+Kubernetes+Versions
916 https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_identifiers.html#identifiers-arns

3. Modify the `mapRoles` and `mapUsers` objects according to the permissions as needed. The example below is mapping the `arn:aws:iam::MYAWSACCOUNTID:role/PowerUser` role to `systems:masters` on the Kubernetes cluster:

```
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::MYAWSACCOUNTID:role/nodes.cluster-api-provider-
aws.sigs.k8s.io
      username: system:node:{{EC2PrivateDNSName}}
    - groups:
      - system:masters
      rolearn: arn:aws:iam::MYAWSACCOUNTID:role/PowerUser
      username: admin
kind: ConfigMap
```

   For more information, consult the Enabling IAM user and role access[917] to you cluster guide and the Kubernetes RBAC guide[918].

4. From your management cluster run the following command to fetch a kubeconfig that uses IAM based permissions by running:

```
dkp get kubeconfig -c ${EKS_CLUSTER_NAME} -n ${KOMMANDER_WORKSPACE_NAMESPACE}
>> ${EKS_CLUSTER_NAME}.conf
```

6.13.4 Grant Cluster Access

Enterprise Gov Advanced

6.13.4.1 How to Grant EKS Cluster Access

You can access your cluster using AWS IAM roles in the dashboard. When you create an EKS cluster, the IAM entity is granted `system:masters` permissions in Kubernetes Role Based Access Control (RBAC) configuration.[919]

917 https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html
918 https://kubernetes.io/docs/reference/access-authn-authz/rbac/
919 https://kubernetes.io/docs/reference/access-authn-authz/rbac/

More information about the configuration of the EKS control plane can be found on the EKS Cluster IAM Policies and Roles (see page 1266) page.

If the EKS cluster was created as a cluster using a self-managed AWS cluster that uses IAM Instance Profiles, you will need to modify the `IAMAuthenticatorConfig` field in the `AWSManagedControlPlane` API object to allow other IAM entities to access the EKS workload cluster. Follow the steps below:

1. Run the following command with your `KUBECONFIG` configured to select the self-managed cluster previously used to create the workload EKS cluster. Ensure you substitute `${CLUSTER_NAME}` and `${CLUSTER_NAMESPACE}` with their corresponding values for your cluster.

   ```
   kubectl edit awsmanagedcontrolplane ${CLUSTER_NAME}-control-plane -n $
   {CLUSTER_NAMESPACE}
   ```

2. Edit the `IamAuthenticatorConfig` field with the IAM Role to the corresponding Kubernetes Role. In this example, the IAM role `arn:aws:iam::111122223333:role/PowerUser` is granted the cluster role `system:masters`. Note that this example uses example AWS resource ARNs[920], so these values should be substituted for real values in the corresponding AWS account.

   ```
   iamAuthenticatorConfig:
       mapRoles:
         - groups:
           - system:bootstrappers
           - system:nodes
           rolearn: arn:aws:iam::111122223333:role/my-node-role
           username: system:node:{{EC2PrivateDNSName}}
         - groups:
           - system:masters
           rolearn: arn:aws:iam::111122223333:role/PowerUser
           username: admin
   ```

For further instructions on changing or assigning `roles` or `clusterroles` to which you can map IAM users or roles, see Amazon Enabling IAM access to your cluster[921].

6.13.4.2 Next Step

EKS Explore your Cluster (see page 1282)

920 https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html
921 https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html

6.13.5 EKS Explore your Cluster

Enterprise Gov Advanced

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster.

Before you start, make sure you have created a workload cluster, as described in Create a New Cluster (see page 1271).

6.13.5.1 Explore the New Kubernetes Cluster

Follow these steps:

1. Get a kubeconfig file for the workload cluster:

   When the workload cluster is created, the cluster lifecycle services generate a kubeconfig file for the workload cluster, and write it to a *Secret*. The kubeconfig file is scoped to the cluster administrator.

   Get the kubeconfig from the *Secret*, and write it to a file, using this command:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

2. List the Nodes using this command:

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
   ```

   Output will be similar to:

   ```
   NAME                                          STATUS   ROLES    AGE   VERSION
   ip-10-0-122-211.us-west-2.compute.internal    Ready    <none>   35m   v1.27.12-
   eks-ae9a62a
   ip-10-0-127-74.us-west-2.compute.internal     Ready    <none>   35m   v1.27.12-
   eks-ae9a62a
   ip-10-0-71-155.us-west-2.compute.internal     Ready    <none>   35m   v1.27.12-
   eks-ae9a62a
   ip-10-0-93-47.us-west-2.compute.internal      Ready    <none>   35m   v1.27.12-
   eks-ae9a62a
   ```

It may take a few minutes for the Status to move to `Ready` while the Pod network is deployed. The Nodes' Status should change to Ready soon after the `calico-node` DaemonSet Pods are Ready.

3. List the Pods using this command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get --all-namespaces pods
```

Output will be similar to:

```
NAMESPACE                NAME                                                READY
STATUS     RESTARTS      AGE
calico-system            calico-kube-controllers-7d6749878f-ccsx9            1/1
Running    0             34m
calico-system            calico-node-2r6l8                                   1/1
Running    0             34m
calico-system            calico-node-5pdlb                                   1/1
Running    0             34m
calico-system            calico-node-n24hh                                   1/1
Running    0             34m
calico-system            calico-node-qrh7p                                   1/1
Running    0             34m
calico-system            calico-typha-7bbcb87696-7pk45                       1/1
Running    0             34m
calico-system            calico-typha-7bbcb87696-t4c8r                       1/1
Running    0             34m
calico-system            csi-node-driver-bz48k                               2/2
Running    0             34m
calico-system            csi-node-driver-k5mmk                               2/2
Running    0             34m
calico-system            csi-node-driver-nvcck                               2/2
Running    0             34m
calico-system            csi-node-driver-x4xnh                               2/2
Running    0             34m
kube-system              aws-node-2xp86                                      1/1
Running    0             35m
kube-system              aws-node-5f2kx                                      1/1
Running    0             35m
kube-system              aws-node-6lzm7                                      1/1
Running    0             35m
kube-system              aws-node-pz8c6                                      1/1
Running    0             35m
kube-system              cluster-autoscaler-789d86b489-sz9x2                 0/1
Init:0/1   0             36m
kube-system              coredns-57ff979f67-pk5cg                            1/1
Running    0             75m
kube-system              coredns-57ff979f67-sf2j9                            1/1
Running    0             75m
kube-system              ebs-csi-controller-5f6bd5d6dc-bplwm                 6/6
Running    0             36m
kube-system              ebs-csi-controller-5f6bd5d6dc-dpjt7                 6/6
Running    0             36m
kube-system              ebs-csi-node-7hmm5                                  3/3
Running    0             35m
```

```
kube-system                 ebs-csi-node-l4vfh                                3/3
Running    0                35m
kube-system                 ebs-csi-node-mfr7c                                3/3
Running    0                35m
kube-system                 ebs-csi-node-v8krq                                3/3
Running    0                35m
kube-system                 kube-proxy-7fc5x                                  1/1
Running    0                35m
kube-system                 kube-proxy-vvkmk                                  1/1
Running    0                35m
kube-system                 kube-proxy-x6hcc                                  1/1
Running    0                35m
kube-system                 kube-proxy-x8frb                                  1/1
Running    0                35m
kube-system                 snapshot-controller-8ff89f489-4cfxv               1/1
Running    0                36m
kube-system                 snapshot-controller-8ff89f489-78gg8               1/1
Running    0                36m
node-feature-discovery      node-feature-discovery-master-7d5985467-52fcn     1/1
Running    0                36m
node-feature-discovery      node-feature-discovery-worker-88hr7               1/1
Running    0                34m
node-feature-discovery      node-feature-discovery-worker-h95nq               1/1
Running    0                35m
node-feature-discovery      node-feature-discovery-worker-lfghg               1/1
Running    0                34m
node-feature-discovery      node-feature-discovery-worker-prc8p               1/1
Running    0                35m
tigera-operator             tigera-operator-6dcd98c8ff-k97hq                  1/1
Running    0                36m
```

6.13.5.1.1 Next Step

EKS Attach a Cluster (see page 1284)

6.13.6 EKS Attach a Cluster

Enterprise Gov Advanced

You can attach existing Kubernetes clusters to the Management Cluster. After attaching the cluster, you can use the UI to examine and manage (see page 774) this cluster. The following procedure shows how to attach an existing Amazon Elastic Kubernetes Service (EKS) cluster.

This procedure assumes you have an existing and spun up Amazon EKS cluster(s) with administrative privileges. Refer to the Amazon EKS[922] for setup and configuration information.

- Install aws-iam-authenticator[923]. This binary is used to access your cluster using kubectl.

6.13.6.1 Attach a Pre-existing EKS Cluster

Ensure that the `KUBECONFIG` environment variable is set to the Management cluster before attaching by running:

```
export KUBECONFIG=<Management_cluster_kubeconfig>.conf
```

6.13.6.1.1 Access your EKS clusters

1. Ensure you are connected to your EKS clusters. Enter the following commands for each of your clusters:

   ```
   kubectl config get-contexts
   kubectl config use-context <context for first eks cluster>
   ```

2. Confirm `kubectl` can access the EKS cluster:

   ```
   kubectl get nodes
   ```

6.13.6.1.2 Create a `kubeconfig` File

To get started, ensure you have kubectl[924] set up and configured with ClusterAdmin[925] for the cluster you want to connect to Kommander.

1. Create the necessary service account:

   ```
   kubectl -n kube-system create serviceaccount kommander-cluster-admin
   ```

922 https://aws.amazon.com/eks/
923 https://docs.aws.amazon.com/eks/latest/userguide/install-aws-iam-authenticator.html
924 https://kubernetes.io/docs/tasks/tools/#kubectl
925 https://kubernetes.io/docs/concepts/cluster-administration/

2. Create a token secret for the `serviceaccount`:

```
kubectl -n kube-system create  -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: kommander-cluster-admin-sa-token
  annotations:
    kubernetes.io/service-account.name: kommander-cluster-admin
type: kubernetes.io/service-account-token
EOF
```

   For more information on Service Account Tokens, refer to this article[926] in our blog.

3. Verify that the `serviceaccount` token is ready by running this command:

```
kubectl -n kube-system get secret kommander-cluster-admin-sa-token -oyaml
```

   Verify that the `data.token` field is populated. The output should be similar to this:

```
apiVersion: v1
data:
  ca.crt: LS0tLS1CRUdJTiBDR...
  namespace: ZGVmYXVsdA==
  token: ZXlKaGJHY2lPaUpTVX...
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: kommander-cluster-admin
    kubernetes.io/service-account.uid: b62bc32e-b502-4654-921d-94a742e273a8
  creationTimestamp: "2022-08-19T13:36:42Z"
  name: kommander-cluster-admin-sa-token
  namespace: default
  resourceVersion: "8554"
  uid: 72c2a4f0-636d-4a70-9f1c-55a75f15e520
type: kubernetes.io/service-account-token
```

4. Configure the new service account for `cluster-admin` permissions:

```
cat << EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kommander-cluster-admin
```

926 https://eng.d2iq.com/blog/service-account-tokens-in-kubernetes-v1.24/#whats-changed-in-kubernetes-v124

```
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kommander-cluster-admin
  namespace: kube-system
EOF
```

5. Set up the following environment variables with the access data that is needed for producing a new kubeconfig file:

```
export USER_TOKEN_VALUE=$(kubectl -n kube-system get secret/kommander-cluster-
admin-sa-token -o=go-template='{{.data.token}}' | base64 --decode)
export CURRENT_CONTEXT=$(kubectl config current-context)
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-
template='{{range .contexts}}{{if eq .name "'''${CURRENT_CONTEXT}'''"}}
{{ index .context "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}"{{with
index .cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{ end }}')
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-
template='{{range .clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}
{{ .cluster.server }}{{end}}{{ end }}')
```

6. Confirm these variables have been set correctly:

```
export -p | grep -E 'USER_TOKEN_VALUE|CURRENT_CONTEXT|CURRENT_CLUSTER|
CLUSTER_CA|CLUSTER_SERVER'
```

7. Generate a kubeconfig file that uses the environment variable values from the previous step:

```
cat << EOF > kommander-cluster-admin-config
apiVersion: v1
kind: Config
current-context: ${CURRENT_CONTEXT}
contexts:
- name: ${CURRENT_CONTEXT}
  context:
    cluster: ${CURRENT_CONTEXT}
    user: kommander-cluster-admin
    namespace: kube-system
clusters:
- name: ${CURRENT_CONTEXT}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
```

```
users:
- name: kommander-cluster-admin
  user:
    token: ${USER_TOKEN_VALUE}
EOF
```

8. This process produces a file in your current working directory called `kommander-cluster-admin-config`. The contents of this file are used in Kommander to attach the cluster.
   Before importing this configuration, verify the `kubeconfig` file can access the cluster:

   ```
   kubectl --kubeconfig $(pwd)/kommander-cluster-admin-config get all --all-namespaces
   ```

6.13.6.2 Attach from UI or Attach Manually Through the CLI

There are two options to attach the cluster. Choose from one of the following based on your preference.

- Through the UI (see page 438)
- Using the DKP CLI (see page 439)

6.13.6.2.1 Finish Attach EKS Cluster from the UI

Now that you have kubeconfig, go to the DKP UI and follow these steps below:

1. From the top menu bar, select your target workspace.
2. On the Dashboard page, select the **Add Cluster** option in the **Actions** dropdown menu at the top right.
3. Select **Attach Cluster**.
4. Select the **No additional networking restrictions** card. Alternatively, if you must use network restrictions, stop following the steps below, and see the instructions on the page Attach a cluster WITH network restrictions (see page 804).
5. Upload the kubeconfig file you created in the previous section (or copy its contents) into the **Cluster Configuration** section.
6. The **Cluster Name** field automatically populates with the name of the cluster in the kubeconfig. You can edit this field with the name you want for your cluster.
7. Add labels to classify your cluster as needed.
8. Select **Create** to attach your cluster.

If a cluster has limited resources to deploy all the federated platform services, it will fail to stay attached in the DKP UI. If this happens, ensure your system has sufficient resources for all pods.

6.13.6.3 Manually Attach a DKP CLI Cluster to the Management Cluster

> ⚠ These steps are only applicable if you do not set a WORKSPACE_NAMESPACE when creating a cluster. If you already set a WORKSPACE_NAMESPACE, then you do not need to perform these steps since the cluster is already attached to the workspace.

Starting with DKP 2.6, when you create a Managed Cluster (see page 80) with the DKP CLI, it attaches automatically to the Management Cluster (see page 80) after a few moments.

However, if you do not set a workspace, the attached cluster will be created in the `default` workspace. To ensure that the attached cluster is created in your desired workspace namespace, follow these instructions:

1. Confirm you have your MANAGED_CLUSTER_NAME variable set with the following command:

   ```
   echo ${MANAGED_CLUSTER_NAME}
   ```

2. Retrieve your kubeconfig from the cluster you have created without setting a workspace:

   ```
   dkp get kubeconfig --cluster-name ${MANAGED_CLUSTER_NAME} > $
   {MANAGED_CLUSTER_NAME}.conf
   ```

3. You can now either [attach it in the UI](link to attaching it to workspace via UI that was earlier), or attach your cluster to the workspace you want in the CLI.
   NOTE: This is only necessary if you never set the workspace of your cluster upon creation.

4. Retrieve the workspace where you want to attach the cluster:

   ```
   kubectl get workspaces -A
   ```

5. Set the WORKSPACE_NAMESPACE environment variable:

   ```
   export WORKSPACE_NAMESPACE=<workspace-namespace>
   ```

6. You need to create a secret in the desired workspace before attaching the cluster to that workspace. Retrieve the kubeconfig secret value of your cluster:

   ```
   kubectl -n default get secret ${MANAGED_CLUSTER_NAME}-kubeconfig -o go-
   template='{{.data.value}}{{ "\n"}}'
   ```

7. This will return a lengthy value. Copy this entire string for a secret using the template below as a reference. Create a new attached-cluster-kubeconfig.yaml file:

```
apiVersion: v1
kind: Secret
metadata:
  name: <your-managed-cluster-name>-kubeconfig
  labels:
    cluster.x-k8s.io/cluster-name: <your-managed-cluster-name>
type: cluster.x-k8s.io/secret
data:
  value: <value-you-copied-from-secret-above>
```

8. Create this secret in the desired workspace:

```
kubectl apply -f attached-cluster-kubeconfig.yaml --namespace $
{WORKSPACE_NAMESPACE}
```

9. Create this `kommandercluster` object to attach the cluster to the workspace:

```
cat << EOF | kubectl apply -f -
apiVersion: kommander.mesosphere.io/v1beta1
kind: KommanderCluster
metadata:
  name: ${MANAGED_CLUSTER_NAME}
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  kubeconfigRef:
    name: ${MANAGED_CLUSTER_NAME}-kubeconfig
  clusterRef:
    capiCluster:
      name: ${MANAGED_CLUSTER_NAME}
EOF
```

10. You can now view this cluster in your Workspace in the UI and you can confirm its status by running the below command. It may take a few minutes to reach "Joined" status:

```
kubectl get kommanderclusters -A
```

If you have several Essential Clusters (see page 0) and want to turn one of them to a Managed Cluster to be centrally administrated by a Management Cluster, refer to Platform Expansion: Convert a DKP Essential Cluster to a DKP Enterprise Managed Cluster (see page 859).

6.13.6.4 Related Information

For information on related topics or procedures, refer to the following:

- Configuring and Running Amazon EKS Clusters[927]
- Manage Clusters (see page 774)

6.13.6.5 Next Step

If needed, you can Delete EKS Cluster from CLI (see page 1291). Otherwise, proceed to Day 2 - Cluster Operations Management (see page 590).

6.13.7 Delete the EKS Cluster from CLI

Enterprise Gov Advanced

> Ensure that the `KUBECONFIG` environment variable is set to the self-managed cluster by running `export KUBECONFIG={SELF_MANAGED_AWS_CLUSTER}.conf`

6.13.7.1 Delete the EKS Cluster

If you prefer to continue working in the terminal or shell using the CLI, the steps for deleting the cluster are listed below. If you are in the DKP UI, you can also delete the cluster from the UI using the steps on this page: Delete EKS Cluster from the DKP UI (see page 1292)

Follow these steps for deletion from the CLI:

1. Ensure your AWS credentials are up to date. If you are using user profiles, then refresh the credentials using the command below. Otherwise, proceed to step 2.

   ```
   dkp update bootstrap credentials aws
   ```

2. Delete the Kubernetes cluster and wait a few minutes:

   Before deleting the cluster, dkp deletes all Services of type LoadBalancer on the cluster. Each Service is backed by an AWS Classic ELB. Deleting the Service deletes the ELB that backs it. To skip this step, use the flag `--delete-kubernetes-resources=false`.

   NOTE: Do not skip this step if the VPC is managed by DKP. When DKP deletes the cluster, it deletes the VPC. If the VPC has any EKS Classic ELBs, EKS does not allow the VPC to be deleted, and DKP cannot delete the cluster.

   ```
   dkp delete cluster --cluster-name=${CLUSTER_NAME}
   ```

927 https://aws.amazon.com/eks/

Output will be similar to:

```
 ✓ Deleting Services with type LoadBalancer for Cluster default/eks-example
 ✓ Deleting ClusterResourceSets for Cluster default/eks-example
 ✓ Deleting cluster resources
 ✓ Waiting for cluster to be fully deleted
Deleted default/eks-example cluster
```

6.13.7.1.1 Known Limitations

NOTE: Be aware of these limitations in the current release of DKP.

- The DKP version used to create the workload cluster must match the DKP version used to delete the workload cluster.

6.13.7.1.2 Next Step

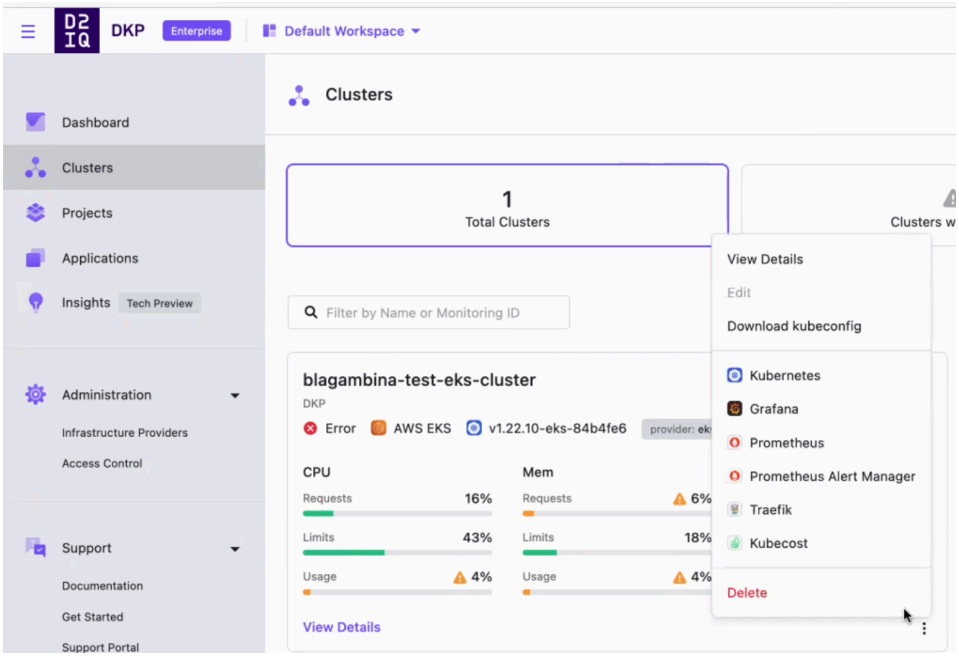
Day 2 - Cluster Operations Management (see page 590)

6.13.7.2 Delete EKS Cluster from the DKP UI

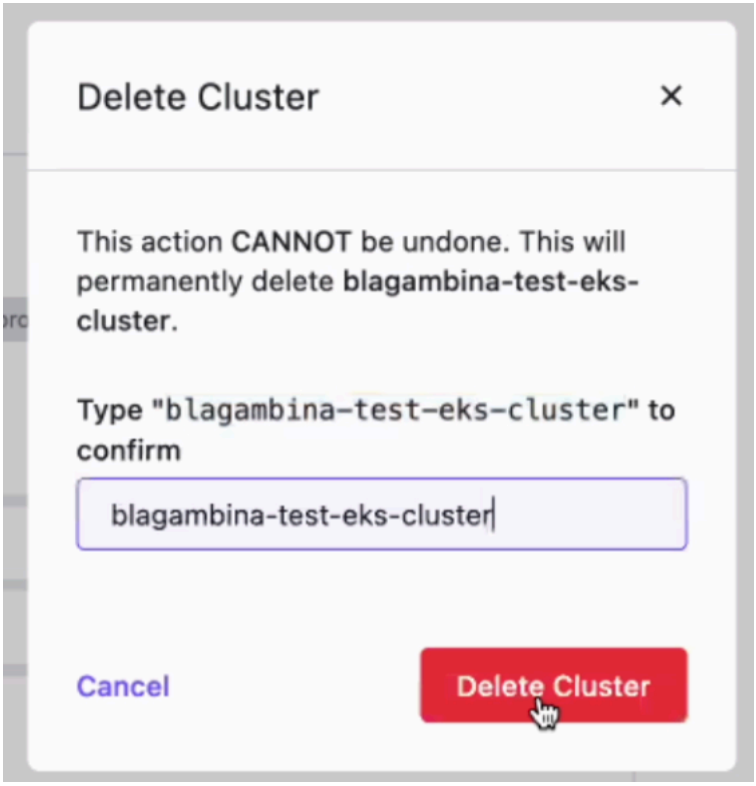
Enterprise Gov Advanced

In order to delete a cluster in the UI, you must first have created a cluster (see page 1278) and have permissions to delete. Otherwise, an administrator privilege could delete a cluster as well.

1. Open the dashboard and select Clusters in the left menu.
2. Select the cluster you wish to delete and click the dotted icon in the bottom right corner.
3. Then select Delete in red.



4. When the next screen appears, copy the name of your cluster and paste it into the empty box.
5. Now execute the deletion using Delete Cluster button.



6. You will see the status as “Deleting” in the top left corner of the cluster you selected for deletion.

This removes the cluster from the DKP UI. For a generic overview of deleting clusters within the UI as well as troubleshooting, see the Disconnect or Delete Clusters (see page 896) instructions.

6.13.8 Manage EKS Nodepools

Enterprise Gov Advanced

Ensure that the `KUBECONFIG` environment variable is set to the self-managed cluster by running `export KUBECONFIG={SELF_MANAGED_AWS_CLUSTER}.conf`

Node pools are part of a cluster and managed as a group, and can be used to manage a group of machines using common properties. New default clusters created by Konvoy contain one node pool of worker nodes that have the same configuration.

You can create additional node pools for specialized hardware or other configurations. For example, if you want to tune your memory usage on a cluster where you need maximum memory for some machines and minimal memory on others, you can create a new node pool with those specific resource needs.

NOTE: Konvoy implements node pools using Cluster API MachineDeployments[928].

6.13.8.1 Create a node pool

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools is to ensure your cluster has nodes deployed in multiple Availability Zones.

By default, the first Availability Zone[929] in the region is used for the nodes in the node pool. To create the nodes in a different Availability Zone set the appropriate `--availability-zone`.

To create a new EKS node pool with 3 replicas, run:

```
dkp create nodepool eks ${NODEPOOL_NAME} \
    --cluster-name=${CLUSTER_NAME} \
    --replicas=3
```

928 https://cluster-api.sigs.k8s.io/developer/architecture/controllers/machine-deployment.html
929 https://aws.amazon.com/about-aws/global-infrastructure/regions_az/

```
machinedeployment.cluster.x-k8s.io/example created
awsmachinetemplate.infrastructure.cluster.x-k8s.io/example created
eksconfigtemplate.bootstrap.cluster.x-k8s.io/example created
 ✓ Creating default/example nodepool resources
```

Advanced users can use a combination of the `--dry-run` and `--output=yaml` flags to get a complete set of node pool objects to modify locally or store in version control.

6.13.8.2 Scaling Up Node Pools

To scale up a node pool in a cluster, run:

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=5 --cluster-name=${CLUSTER_NAME}
```

Your output should be similar to this example, indicating the scaling is in progress:

```
✓ Scaling node pool example to 5 replicas
```

After a few minutes, you can list the node pools to:

```
dkp get nodepools --cluster-name=${CLUSTER_NAME}
```

Your output should be similar to this example, with the number of DESIRED and READY replicas increased to 5:

```
NODEPOOL                        DESIRED             READY
KUBERNETES VERSION
example                         5                   5                   v1.23.6
eks-example-md-0                4                   4                   v1.23.6
```

6.13.8.3 Delete EKS Node Pools

Delete node pools in a cluster

Deleting a node pool deletes the Kubernetes nodes and the underlying infrastructure. All nodes are drained prior to deletion and the pods running on those nodes are rescheduled.

To delete a node pool from a managed cluster, run:

```
dkp delete nodepool ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME}
```

Here, `example` is the node pool to be deleted.

The expected output will be similar to the following example, indicating the node pool is being deleted:

```
✓ Deleting default/example nodepool resources
```

Deleting an invalid node pool results in output similar to this example:

```
dkp delete nodepool ${CLUSTER_NAME}-md-invalid --cluster-name=${CLUSTER_NAME}
```

```
MachineDeployments or MachinePools.infrastructure.cluster.x-k8s.io "no
MachineDeployments or MachinePools found for cluster eks-example" not found
```

6.14 Azure Infrastructure

This section describes how you create a DKP cluster in an Azure environment. Start with the prerequisites and follow the links at the bottom of the page to proceed through the entire process.

- Azure Prerequisites (see page 1296)
- Azure Cluster Creation Customization Choices (see page 1302)
- Azure Install in a Non-air-gapped Environment (see page 1311)
- Azure Management Tools (see page 1333)

6.14.1 Azure Prerequisites

Prepare your machine and environment to run DKP

6.14.1.1 DKP Prerequisites

Before you begin using DKP you must have:

- An x86_64-based Linux or macOS machine.
- Download (see page 76) the `dkp` binary for Linux, or macOS. To check which version of DKP you installed for compatibility reasons, run the `dkp version -h` command (dkp version (see page 1943)).
- A Container engine/runtime installed is required to install DKP:
  - Version Docker®[930] container engine version 18.09.2 or higher installed for Linux or MacOS - On macOS, Docker runs in a virtual machine which needs configured with at least 8 GB of memory.
  - Version 4.0 of Podman[931] or higher for Linux. Host requirements found here: Host Requirements[932]
- kubectl[933] for interacting with the running cluster.

930 https://docs.docker.com/get-docker/
931 https://podman.io/getting-started/installation
932 https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
933 https://kubernetes.io/docs/tasks/tools/#kubectl

- Azure CLI[934].
- A valid Azure account with credentials configured[935].
- Create a custom Azure image using KIB (see page 1626).

On macOS, Docker runs in a virtual machine. Configure this virtual machine with at least 8GB of memory.

6.14.1.1.1 Control plane nodes

You should have at least three control plane nodes. Each control plane node should have at least:

- 4 cores
- 16 GiB memory
- Approximately 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.
- Disk usage must be below 85% on the root volume.

DKP on Azure defaults to deploying a `Standard_D4s_v3` virtual machine with an 128 GiB volume for the OS and an 80GiB volume for etcd storage, which meets the above requirements.

6.14.1.1.2 Worker nodes

You should have at least four worker nodes. The specific number of worker nodes required for your environment can vary depending on the cluster workload and size of the nodes. Each worker node should have at least:

- 8 cores
- 32 GiB memory
- Around 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.
- Disk usage must be below 85% on the root volume.

DKP on Azure defaults to deploying a `Standard_D8s_v3` virtual machine with an 80 GiB volume for the OS, which meets the above requirements.

If you use these instructions to create a cluster on Azure using the DKP default settings without any edits to configuration files or additional flags, your cluster is deployed on an Ubuntu 20.04 operating system image (see page 67) with 3 control plane nodes, and 4 worker nodes which match the requirements above.

934 https://docs.microsoft.com/en-us/cli/azure/install-azure-cli

935 https://github.com/kubernetes-sigs/cluster-api-provider-azure/blob/master/docs/book/src/topics/getting-started.md#prerequisites

6.14.1.2 Azure Prerequisites

In Azure, application registration, application objects, and service principals in Azure Active Directory[936] (Azure AD) are used for access. To delegate identity and access management functions to Azure AD, an **application** must be registered with an Azure AD tenant. An Azure AD application is defined by its one and only **application object**, which resides in the Azure AD. To access resources that are secured by an Azure AD tenant, the entity that requires access must be represented by a security principal. This requirement is true for both users (user principal) and applications (**service principal**). Therefore, a service principal is a prerequisite and the next step explains it.

6.14.1.2.1 Next Step

Azure Create a Service Principal (see page 1298)

6.14.1.3 Azure Create a Service Principal

An Azure service principal is an identity created for use with applications, hosted services and other automated tools used to access resources in Azure. Service principals[937] provide access to Azure resources with your subscription level. The access is restricted by the roles[938] assigned to the service principal.

6.14.1.3.1 Configure Azure Service Principal

> If you have already set a service principal, then the environment variables needed by KIB (`[AZURE_CLIENT_SECRET`, `AZURE_CLIENT_ID`, `AZURE_TENANT_ID`, `AZURE_SUBSCRIPTION_ID]`) are set and do not need repeated if you are still working in the same window.

If you have not executed the Azure Prerequisite (see page 1296) steps, they are listed below.

1. Sign in to Azure:

```
az login
```

```
[
```

936 https://learn.microsoft.com/en-us/azure/active-directory/develop/app-objects-and-service-principals?tabs=browser
937 https://learn.microsoft.com/en-us/azure/active-directory/develop/app-objects-and-service-principals?tabs=browser
938 https://learn.microsoft.com/en-us/azure/databricks/security/auth-authz/access-control/service-principal-acl

```
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "id": "b1234567-abcd-11a1-a0a0-1234a5678b90",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Mesosphere Developer Subscription",
    "state": "Enabled",
    "tenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "user": {
      "name": "user@azuremesosphere.onmicrosoft.com",
      "type": "user"
    }
  }
]
```

2. Create an Azure Service Principal (SP) by running the following command:

   If an SP with the name exists, this command will rotate the password.

```
az ad sp create-for-rbac --role contributor --name "$(whoami)-konvoy" --
scopes=/subscriptions/$(az account show --query id -o tsv) --query
"{ client_id: appId, client_secret: password, tenant_id: tenant }"
```

```
{
  "client_id": "7654321a-1a23-567b-b789-0987b6543a21",
  "client_secret": "Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C",
  "tenant_id": "a1234567-b132-1234-1a11-1234a5678b90"
}
```

3. Set the `AZURE_CLIENT_SECRET` environment variable:

```
export AZURE_CLIENT_SECRET="<azure_client_secret>" #
Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C
export AZURE_CLIENT_ID="<client_id>"               # 7654321a-1a23-567b-
b789-0987b6543a21
export AZURE_TENANT_ID="<tenant_id>"               # a1234567-b132-1234-1a11-12
34a5678b90
export AZURE_SUBSCRIPTION_ID="<subscription_id>"   # b1234567-abcd-11a1-
a0a0-1234a5678b90
```

4. Ensure you have an override file (see page 1670) to configure specific attributes of your Azure image. Otherwise, edit the YAML file[939] for your OS directly.

939 https://github.com/mesosphere/konvoy-image-builder/tree/7447074a6d910e71ad2e61fc4a12820d073ae5ae/images/azure

6.14.1.3.1.1 Next Step

Azure Using Konvoy Image Builder (see page 1300)

6.14.1.4 Azure Using Konvoy Image Builder

This procedure describes how to use the Konvoy Image Builder (see page 1626) (KIB) to create a Cluster API[940] compliant Azure Virtual Machine (VM) Image. The VM Image contains the base operating system you specify and all the necessary Kubernetes components. The Konvoy Image Builder uses variable `overrides` to specify the base image and container images to use in your new Azure VM image.

The default Azure image is not recommended for use in production. We suggest using KIB for Azure to build the image in order to take advantage of enhanced cluster operations. Explore the Customize your Image (see page 1661) topic for more options.
For more information regarding using the image in creating clusters, refer to the Azure Create a New Cluster (see page 1314) section of the documentation.

6.14.1.4.1 Prerequisites

Before you begin, you must:

- Download the Konvoy Image Builder (see page 1626) bundle for your version of DKP.
- Check the Supported Infrastructure Operating Systems (see page 67)
- Check the Supported Kubernetes Version (see page 1706) for your Provider.
- Create a working `Docker` setup.

6.14.1.4.2 Extract the KIB Bundle

Extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION_$OS` folder. The bundled version of `konvoy-image` contains an embedded `docker` image that contains all the requirements for building.

The `konvoy-image` binary and all supporting folders are also extracted. When extracted, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

940 https://cluster-api.sigs.k8s.io/

Ensure you have created a Service Principal (see page 1298)and then create your custom image using KIB.

6.14.1.4.3 Build the Image

Run the `konvoy-image` command to build and validate the image.

```
konvoy-image build azure --client-id ${AZURE_CLIENT_ID} --tenant-id $
{AZURE_TENANT_ID} --overrides override-source-image.yaml images/azure/ubuntu-2004.yam
l
```

By default, the image builder builds in the `westus2` location. To specify another location set the `--location` flag (shown in example below is how to change the location to `eastus`):

```
konvoy-image build azure --client-id ${AZURE_CLIENT_ID} --tenant-id $
{AZURE_TENANT_ID} --location eastus --overrides override-source-image.yaml images/
azure/centos-7.yaml
```

When the command is complete, the image id is printed and written to the `./packer.pkr.hcl` file. This file has an `artifact_id` field whose value provides the name of the image. You should then specify this image id when creating the cluster.

6.14.1.4.4 Image Gallery

By default Konvoy Image Builder will create a Resource Group, Gallery, and Image Name to store the resulting image in. To specify a specific Resource Group, Gallery, or Image Name flags may be specified:

```
--gallery-image-locations string     a list of locations to publish the image
(default same as location)
--gallery-image-name string          the gallery image name to publish the image to
--gallery-image-offer string         the gallery image offer to set (default "dkp")
--gallery-image-publisher string     the gallery image publisher to set (default
"dkp")
--gallery-image-sku string           the gallery image sku to set
--gallery-name string                the gallery name to publish the image in
(default "dkp")
--resource-group string              the resource group to create the image in
(default "dkp")
```

When creating your cluster, you will then add this flag during the create process for your custom image: `--compute-gallery-id "<Managed Image Shared Image Gallery Id>"`. See Create a New Azure Cluster (see page 1314) for specific consumption of image commands.

The SKU and Image Name will default to the values found in the image YAML.

Ensure you have named the correct YAML file for your OS[941] in the `konvoy-image build` command.

Marketplace Images for Rocky Linux

Similar to Image Gallery, additional flags allow DKP to create a cluster with Marketplace based images for Rocky Linux 9.0: `--plan-offer`, `--plan-publisher` and `--plan-sku`.

- If you use these fields in the override file (see page 1670) when you create a machine image with KIB, you must also set the corresponding flags when you create your cluster with DKP.
- Conversely, if you do not use these fields when you create a machine image with KIB, you do not need to set these flags when you create your cluster with DKP.

```
---
download_images: true

packer:
  distribution: "rockylinux-9"                # Offer
  distribution_version: "rockylinux-9"  # SKU
  # Azure Rocky linux official image: https://portal.azure.com/#view/
Microsoft_Azure_Marketplace/GalleryItemDetailsBladeNopdl/id/
erockyenterprisesoftwarefoundationinc1653071250513.rockylinux-9
  image_publisher: "erockyenterprisesoftwarefoundationinc1653071250513"
  image_version: "latest"
  ssh_username: "azureuser"
  plan_image_sku: "rockylinux-9" # SKU
  plan_image_offer: "rockylinux-9" # offer
  plan_image_publisher: "erockyenterprisesoftwarefoundationinc1653071250513" #
publisher

build_name: "rocky-90-az"
packer_builder_type: "azure"
python_path: ""
```

6.14.1.4.5 Next Step

Azure Cluster Creation Customization Choices (see page 1302)

6.14.2 Azure Cluster Creation Customization Choices

Below are a some methods to customize your cluster during creation. If none of these choices apply, skip this page and proceed to

- Azure Install in a Non-air-gapped Environment (see page 1311)

941 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/gcp

6.14.2.1 Section Topics

When creating clusters, many options are available such as those listed in this section of the documentation. Familiarize yourself with the flags required to apply these customizations during cluster creation.

- Azure Customizing CAPI Clusters (see page 1303) — Familiarize yourself with Cluster API before editing the cluster objects because edits can prevent the cluster from deploying successfully.
- Azure Registry Mirrors (see page 1304) — Configure your cluster to use an existing local registry when attempting to pull images by adding the flag(s) to the dkp create cluster command to pull images from your local registry.
- Azure Load the Registry (see page 1305) — Because air-gapped environments do not have direct access to the Internet, you must download, extract and load several required images to your local container registry, before installing DKP.
- Azure HTTP Proxy (see page 1308) — When creating a DKP cluster in environments that use an HTTP/HTTPS proxy, you must provide proxy details. The proxy values are strings that list a set of proxy servers, URLs, or wildcard addresses that is specific to your environment.
- Azure Output Directory YAML (see page 1309) — You can create individual files with different smaller manifests for ease in editing using the --output-directory flag used with --output=json|yaml. You create the directory of where to output resources to files.
- Azure Custom DNS (see page 1310) — To use a custom Domain Name Servers(DNS) on Azure, you need a DNS name in your control. Once the resource group has been created, you can create your hosted zone with the command below:
- Azure Marketplace (see page 1310) — To allow DKP to create a cluster with Marketplace based images such as for Rocky Linux, you must specify them with flags.

6.14.2.1.1 Next Step

After you make decisions about customization choices, proceed to the environment instructions:

- Azure Install in a Non-air-gapped Environment (see page 1311)

6.14.2.2 Azure Customizing CAPI Clusters

Familiarize yourself with Cluster API (see page 1061) before editing the cluster objects because edits can prevent the cluster from deploying successfully.

The result of this command will allow such edits:

```
dkp create cluster azure \
    --cluster-name=${CLUSTER_NAME} \
    --dry-run \
    --output=yaml \
  > ${CLUSTER_NAME}.yaml
```

To edit the YAML, you need to understand the CAPI components to avoid breaking the cluster. For expanded component explanation, refer to the Universal Configurations for All Providers section of the documentation in the section: Customizing CAPI Components for a Cluster (see page 1061)

6.14.2.2.1 Next Topic

Azure Registry Mirrors (see page 1304)

6.14.2.3 Azure Registry Mirrors

Configure your cluster to use an existing local registry (see page 1606) when attempting to pull images by adding the flag(s) to the `dkp create cluster` command to pull images from your local registry.

Kubernetes does not natively provide a registry for hosting the container images you will use to run the applications you want to deploy on Kubernetes. Instead, Kubernetes requires you to use an external solution for storing and sharing container images. There are a variety of Kubernetes-compatible registry options that are compatible with DKP.

6.14.2.3.1 How Does it Work?

The **first time** you request an image from your local registry mirror, it pulls the image from the public registry (such as Docker) and stores it locally before handing it back to you. On subsequent requests, the local registry mirror is able to serve the image from its own storage.

6.14.2.3.2 Air-gapped vs Non-air-gapped Environments

In a non-air-gapped environment, you have access to the Internet. You retrieve artifacts from specialized repositories dedicated to them, such as Docker images contained in DockerHub and Helm Charts that come from a dedicated Helm Chart repository. You can also create your own local repository to hold the downloaded container images needed or any custom images you've created with the Konvoy Image Builder (see page 1626) tool.

In an **air-gapped environment**, you need a local repository to store Helm charts, Docker images, and other artifacts. Private registries provide security and privacy into enterprise container image storage, whether hosted remotely or on-premises locally in an air-gapped environment. DKP in an air-gapped environment **requires** a local container registry of trusted images to enable production-level Kubernetes cluster management. However, a local registry is an option in a non-air-gapped environment as well for speed and security.

If you want to use images from this local registry to deploy applications inside your Kubernetes cluster, you'll need to set up a **secret** for a private registry. The secret contains your login data, which Kubernetes needs to connect to your private repository.

Set the environment variable with your registry information.

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_USERNAME=<username>
```

```
export REGISTRY_PASSWORD=<password>
export REGISTRY_CA=<path to the cacert file on the bastion>
```

Definitions:

- `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.

Other local registries may use the options below:

- JFrog - `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional-set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.
- To increase Docker Hub's rate limit[942] use your Docker Hub credentials when creating the cluster, by setting flags `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username=<your-username> --registry-mirror-password=<your-password>` when running `dkp create cluster` .

Use the flag now during `dkp create cluster` : `--registry-mirror-url`

6.14.2.3.2.1 Related Topics

More information and detail can be found:

- Registry Mirror Tools (see page 1606)
- Use a Registry Mirror (see page 1609)

6.14.2.3.2.2 Next Step

If using a Registry Mirror, you need to load the images with Azure Load the Registry (see page 1305).

6.14.2.4 Azure Load the Registry

Because air-gapped environments do not have direct access to the Internet, you must download, extract and load several required images to your local container registry, before installing DKP.

If desired, environments that are non-air-gapped can also perform the follow steps to use a local registry for speed and security reasons.

6.14.2.4.1 Load Images into your Registry

This page assumes you have already downloaded (see page 76) and extracted (see page 1211) the bundle from the Prerequisites (see page 1151). The sections below explain further how you are pushing the images to your registry and then using them in creating a cluster.

942 https://docs.docker.com/docker-hub/download-rate-limit/

6.14.2.4.2 Download all Images for Air-gapped Deployments

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for prerequisites to download and then how to push the necessary images to this registry.

1. Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`) to load registry images as explained below.
2. Connectivity with clusters attaching to the management cluster is required:
   - Both management and attached clusters must be able to connect to the local registry.
   - The management cluster must be able to connect to all attached cluster's API servers.
   - The management cluster must be able to connect to any load balancers created for platform services on the management cluster.

6.14.2.4.3 Extract Air-gapped Images and Set Variables

Follow these steps to **extract** the air-gapped image bundles into your private registry:

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

   ```
   cd dkp-v2.8.1
   ```

3. Set an environment variable with your registry address:

   ```
   export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   ```

- `REGISTRY_URL`: the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- NOTE: Other local registries may use the options below:
  - JFrog - `CONTAINER_REGISTRY_CA`: (optional) the path on the bastion machine to the registry CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.

- `CONTAINER_REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
- `CONTAINER_REGISTRY_PASSWORD` : optional if username is not set.

6.14.2.4.3.1 Load Images to your Private Registry - Konvoy

Before creating or upgrading a Kubernetes cluster, you need to load the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, refer to Local Registry Tools (see page 1606) page for more information.

Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

6.14.2.4.3.2 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

6.14.2.4.3.3 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME}
--to-registry-password=${REGISTRY_PASSWORD}
```

For more information to set up a private registry with a registry mirror, see this page (see page 1609) for details on using that flag.

6.14.2.4.3.4 Next Topic

Azure HTTP Proxy (see page 1308)

6.14.2.5 Azure HTTP Proxy

When creating a DKP cluster in environments that use an HTTP/HTTPS proxy, you must provide proxy details. The proxy values are strings that list a set of proxy servers, URLs, or wildcard addresses that is specific to your environment.

If your environment uses HTTP/HTTPS proxies, you must include the flags and their related values in commands for the proxy to be successful throughout various steps of installation:

- `--http-proxy`
- `--https-proxy`
- `--no-proxy`

To create a proxied environment, you need to include flags at these action item points:

- Bootstrap cluster
- CAPI components
- Cluster creation
- DKP Kommander component

Create the bootstrap cluster and CAPI components using the appropriate commands, `dkp create bootstrap` and `dkp create capi-components` respectively, combined with the command line flags to include your HTTP/S proxy information.

You can also specify HTTP/S proxy information in an override file when using Konvoy Image Builder (KIB) (see page 1626).

Without these values provided as part of the relevant `dkp create` command, DKP cannot create the requisite parts of your new cluster correctly. This is true of both management and managed clusters (see page 79) alike.

For full HTTP Proxy configuration, you need to specify proxy settings using all the details in the Configuring an HTTP/HTTPS Proxy (see page 1053)section of the documentation for:

- Configure the Bootstrap Cluster HTTP Proxy Settings (see page 1054)

- Create CAPI Components with HTTP/HTTPS Proxy Settings (see page 1055)
- Create a Cluster with HTTP/HTTPS Proxy (see page 1056)
- Configure an HTTP/HTTPS Proxy for the DKP Kommander Component (see page 1058)

6.14.2.5.1 HTTP Proxy Cluster Example

```
dkp create cluster azure \
 --cluster-name ${CLUSTER_NAME} \
 --control-plane-http-proxy="${CONTROL_PLANE_HTTP_PROXY}" \
 --control-plane-https-proxy="${CONTROL_PLANE_HTTPS_PROXY}" \
 --control-plane-no-proxy="${CONTROL_PLANE_NO_PROXY}" \
 --worker-http-proxy="${WORKER_HTTP_PROXY}" \
 --worker-https-proxy="${WORKER_HTTPS_PROXY}" \
 --worker-no-proxy="${WORKER_NO_PROXY}"
```

6.14.2.5.2 Next Topic

Azure Output Directory YAML (see page 1309)

6.14.2.6 Azure Output Directory YAML

You can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag used with `--output=json|yaml`. You create the directory of where to output resources to files.

Using this flag will create multiple files in the specified directory which must already exist:

```
dkp create cluster azure
    --cluster-name=${CLUSTER_NAME} \
    --dry-run \
    --output=yaml \
    --output-directory=<existing-directory>
```

For more information regarding this flag or others, please refer to the CLI section of the documentation for the dkp create cluster (see page 1853) command and select your provider.

6.14.2.6.1 Next Topic

Azure Custom DNS (see page 1310)

6.14.2.7 Azure Custom DNS

To use a custom Domain Name Servers(DNS) on Azure, you need a DNS name in your control. Once the resource group has been created, you can create your hosted zone with the command below:

```
az network dns zone create --resource-group "d2iq-professional-services" --name
```

You no longer need to create a cluster issuer. There are several documents that explain custom DNS (see page 1569) in the Kommander component. For more information from the Azure site, refer to their DNS Overview[943].

6.14.2.7.1 Next Topic

Azure Marketplace (see page 1310)

6.14.2.8 Azure Marketplace

To allow DKP to create a cluster with Marketplace based images such as for Rocky Linux, you must specify them with flags.

If these fields were specified in the override file (see page 1670) during image creation (see page 1642), the flags must be used in cluster creation:

- `--plan-offer`, `--plan-publisher` and `--plan-sku`
- ```
  --plan-offer rockylinux-9
  --plan-publisher erockyenterprisesoftwarefoundationinc1653071250513
  --plan-sku rockylinux-9
  ```

If you see a similar error to "Creating a virtual machine from Marketplace image or a custom image sourced from a Marketplace image requires Plan information in the request." when creating a cluster, you must also set the following flags `--plan-offer`, `--plan-publisher`, `--plan-sku`. For example when creating a cluster with Rocky Linux VMs, add the following flags to your `dkp create cluster azure` command:

- `--plan-offer`, `--plan-publisher` and `--plan-sku`

See Azure Marketplace[944] and Rocky Linux[945] for images.

943 https://learn.microsoft.com/en-us/azure/dns/dns-overview
944 https://azuremarketplace.microsoft.com/en-us/marketplace/apps?search=procomputers&page=1
945 https://forums.rockylinux.org/t/azure-rocky-image-on-marketplace/5230
  ```

6.14.2.8.1 Next Step

After deciding on any customizations, proceed toAzure Install in a Non-air-gapped Environment (see page 1311).

6.14.3 Azure Install in a Non-air-gapped Environment

This section provides instructions to install DKP in an Azure non-air-gapped environment with custom settings. First you create an Bootstrap Cluster (see page 1187) and then move the CAPI resources to the workload cluster and delete the bootstrap cluster.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

6.14.3.1 Azure Specific Prerequisites

Before you begin using Konvoy with Azure, ensure you met all the prerequisites (see page 1296).

> To deploy a cluster with a custom image in a region where CAPI images[946] are not provided, you need to use Konvoy Image Builder (see page 1153) to create your own image for the region.

6.14.3.1.1 Next Step

Azure Bootstrap (see page 1311)

6.14.3.2 Azure Bootstrap

6.14.3.2.1 Prepare to deploy Kubernetes clusters

To create Kubernetes clusters, Konvoy uses Cluster API[947] (CAPI) controllers. These controllers run on a Kubernetes cluster. To get started, you need a *bootstrap* cluster. By default, Konvoy creates a bootstrap cluster for you in a Docker container using the Kubernetes-in-Docker (KIND[948]) tool.

946 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html
947 https://cluster-api.sigs.k8s.io/
948 https://github.com/kubernetes-sigs/kind

6.14.3.2.2 Prerequisites

Before you begin, you must:

- Complete the steps in Prerequisites (see page 1296).
- Ensure the `dkp` binary can be found in your $PATH.

6.14.3.2.2.1 Bootstrap Cluster Lifecycle Services

1. Review Universal Configurations for all Infrastructure Providers (see page 1052) regarding settings, flags and other choices and then begin bootstrapping.
2. Create a bootstrap cluster:

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config
   ```

   - Configuring an HTTP/HTTPS Proxy (see page 1053) use `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful.

HTTP Flags if needed:

To create a bootstrap cluster in a proxied environment (see page 1054) use this command syntax, in addition to any other flags you may need:

```
dkp create bootstrap --kubeconfig $HOME/.kube/config \
  --http-proxy <string> \
  --https-proxy <string> \
  --no-proxy <string>
```

Konvoy creates a bootstrap cluster using KIND[949] as a library. Konvoy then deploys the following Cluster API[950] providers on the cluster:

- Core Provider[951]
- Azure Infrastructure Provider[952]
- kubeadm Bootstrap Provider[953]
- kubeadm ControlPlane Provider[954]

Konvoy waits until the controller-manager and webhook deployments of these providers are ready. List these deployments using this command:

949 https://github.com/kubernetes-sigs/kind
950 https://cluster-api.sigs.k8s.io/
951 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/
952 https://github.com/kubernetes-sigs/cluster-api-provider-azure
953 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/bootstrap/kubeadm
954 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/controlplane/kubeadm

```
kubectl get --all-namespaces deployments -l=clusterctl.cluster.x-k8s.io
```

```
NAMESPACE                           NAME
READY   UP-TO-DATE   AVAILABLE   AGE
capa-system                         capa-controller-manager                         1
/1      1            1           69s
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-manager       1
/1      1            1           71s
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-manager   1
/1      1            1           70s
capi-system                         capi-controller-manager                         1
/1      1            1           73s
cappp-system                        cappp-controller-manager                        1
/1      1            1           66s
capv-system                         capv-controller-manager                         1
/1      1            1           65s
capz-system                         capz-controller-manager                         1
/1      1            1           67s
cert-manager                        cert-manager                                    1
/1      1            1           16m
cert-manager                        cert-manager-cainjector                         1
/1      1            1           16m
cert-manager                        cert-manager-webhook                            1
/1      1            1           16m
```

6.14.3.2.2.2 (Optional) Create Identity Secret for Azure

If your bootstrap cluster resides on a Virtual machine inside Azure, create an identity secret that uses the cappz-controller:

```
export AZURE_CLUSTER_IDENTITY_SECRET_NAME="cluster-identity-secret"
export CLUSTER_IDENTITY_NAME="cluster-identity"
export AZURE_CLUSTER_IDENTITY_SECRET_NAMESPACE="default"

kubectl create secret generic ${AZURE_CLUSTER_IDENTITY_SECRET_NAME} --from-
literal=clientSecret=${AZURE_CLIENT_SECRET}
```

6.14.3.2.2.3 Next Step

Azure Create a New Cluster (see page 1314)

6.14.3.3 Azure Create a New Cluster

6.14.3.3.1 Prerequisites

- Before you begin, make sure you have created a Bootstrap (see page 1311) cluster.

6.14.3.3.1.1 Name your Cluster

1. Give your cluster a unique name suitable for your environment.
2. Set the environment variable:

```
export CLUSTER_NAME=<azure-example>
```

   a. To create a cluster name that is unique, use the following command. This creates a unique name every time you run it, so use it with forethought:

```
export CLUSTER_NAME=azure-example-$(LC_CTYPE=C tr -dc 'a-z0-9' </dev/
urandom | fold -w 5 | head -n1)
echo $CLUSTER_NAME
```

```
azure-example-pf4a3
```

6.14.3.3.1.2 Important Notes for Azure Environments

1. To use a custom **Azure Image** when creating your cluster, you must create that Azure Image using KIB (see page 1642) first and then use the flag `--compute-gallery-id` to apply the image.

```
...
--compute-gallery-id "<Managed Image Shared Image Gallery Id>"
```

Important to remember: The `--compute-gallery-id` image will be in the format `--compute-gallery-id /subscriptions/<subscription id>/resourceGroups/<resource group name>/providers/Microsoft.Compute/galleries/<gallery name>/images/<image definition name>/versions/<version id>`.

2. Ensure your subnets (see page 1065) do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the Kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

3. **Availability zones (AZs)** are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools to ensure your cluster has nodes deployed in multiple Availability Zones.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single Availability Zone. You may create additional node pools in other Availability Zones with the `dkp create nodepool` command. See Microsoft's documentation for more information on Availability Options for Azure VM[955].

4. **Default Storage Provisioning**: The below cluster creation directions instead describes how to create a cluster using Azure as the infrastructure provider provisioning clusters, which uses Azure Disks Container Storage Interface as the default StorageClass.

If you are using Azure as a Pre-provisioned environment: DKP uses `localvolumeprovisioner` as the default storage provider (see page 110) if creating a pre-provisioned Azure cluster (see page 1079). However, `localvolumeprovisioner` is not suitable for production use. You should use a Kubernetes CSI[956] compatible storage that is suitable for production.

955 https://docs.microsoft.com/en-us/azure/virtual-machines/manage-availability#configure-multiple-virtual-machines-in-an-availability-set-for-redundancy

956 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

You can choose from any of the storage options[957] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass `localvolumeprovisioner` as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[958].

6.14.3.3.1.3 Create a New Azure Kubernetes Cluster

1. Generate the Kubernetes cluster objects. See dkp create cluster azure (see page 1859) reference for the full list of cluster creation options.
2. **(Optional)** Use a local registry mirror (see page 1609). Configure your cluster to use an existing local registry (see page 1606) as a mirror when attempting to pull images previously pushed to your registry (see page 1305).

Export Registry Variables

If you have a local registry, you must provide additional arguments when creating the cluster. These tell the cluster where to locate the local registry to use by defining the URL. Set the needed environment variable(s) with your registry information:

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_CA=<path to the CA on the bastion>
export REGISTRY_USERNAME=<username>
export REGISTRY_USERNAME=<password>
```

- `REGISTRY_URL` : the address of an existing container registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- `REGISTRY_CA` : (optional) the path on the bastion machine to the container registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.

When **creating the cluster**, apply the variables you defined above during the `dkp create cluster` command with the flags needed for your environment:

```
--registry-mirror-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD}
```

See also: Azure Registry Mirrors (see page 1304)

957 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
958 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

3. Create a cluster:

```
dkp create cluster azure \
--cluster-name=${CLUSTER_NAME} \
--dry-run \
--output=yaml \
> ${CLUSTER_NAME}.yaml
```

Expand the drop-downs for **HTTP, FIPS and other flags** to use during the cluster creation step above. For more information regarding this flag or others, please refer to the CLI section of the documentation for dkp create cluster (see page 1853) and select your provider.

FIPS Requirements

To create a cluster in FIPS mode (see page 1598), inform the controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes by adding those flags to `dkp create cluster` command:

```
--kubernetes-version=v1.28.7+fips.0 \
--etcd-version=3.5.10+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
```

HTTP ONLY

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

```
--http-proxy <<http proxy list>>
--https-proxy <<https proxy list>>
--no-proxy <<no proxy list>>
```

- To configure the Control Plane and Worker nodes to use an HTTP proxy:

```
export CONTROL_PLANE_HTTP_PROXY=http://example.org:8080
export CONTROL_PLANE_HTTPS_PROXY=http://example.org:8080
export
CONTROL_PLANE_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1
,10.96.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.s
vc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.sv
c.cluster,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"

export WORKER_HTTP_PROXY=http://example.org:8080
```

```
export WORKER_HTTPS_PROXY=http://example.org:8080
export
WORKER_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96.
0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kube
rnetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.clust
er,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"
```

- Replace:
  - `example.org,example.com,example.net` with you internal addresses
  - `localhost` and `127.0.0.1` addresses should not use the proxy
  - `10.96.0.0/12` is the default Kubernetes service subnet
  - `192.168.0.0/16` is the default Kubernetes pod subnet
  - `kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local` is the internal Kubernetes kube-apiserver service
  - `.svc,.svc.cluster,.svc.cluster.local` is the internal Kubernetes services
- Create a Kubernetes cluster with HTTP proxy configured by adding these flags to the `dkp create cluster` command:

```
--control-plane-http-proxy=${CONTROL_PLANE_HTTP_PROXY} \
--control-plane-https-proxy=${CONTROL_PLANE_HTTPS_PROXY} \
--control-plane-no-proxy=${CONTROL_PLANE_NO_PROXY} \
--worker-http-proxy=${WORKER_HTTP_PROXY} \
--worker-https-proxy=${WORKER_HTTPS_PROXY} \
--worker-no-proxy=${WORKER_NO_PROXY} \
```

Custom DNS

To use a custom DNS on Azure (see page 1310), you need a DNS name in your control. Once the resource group has been created, you can create your hosted zone with the command below:

```
az network dns zone create --resource-group "d2iq-professional-services" --name
```

You no longer need to create a cluster issuer. There are several documents that explain custom DNS (see page 1569) in the Kommander component.

Using Custom Image

When creating your cluster, you will add this flag during the create process for your custom image: `--compute-gallery-id "<Managed Image Shared Image Gallery Id>"`. See the Prerequisites section Azure Using Konvoy Image Builder (see page 1300) for specific consumption of image commands.

The SKU and Image Name will default to the values found in the image YAML. Ensure you have named the correct YAML file for your OS[959] in the `konvoy-image build` command.

Marketplace Image - Rocky Linux

To allow DKP to create a cluster with Marketplace based images such as for Rocky Linux, the following flags are available. If these fields were specified in the override file (see page 1670) during image creation (see page 1642), the flags must be used in cluster creation:

- `--plan-offer`, `--plan-publisher` and `--plan-sku`

```
--plan-offer rockylinux-9
--plan-publisher erockyenterprisesoftwarefoundationinc1653071250513
--plan-sku rockylinux-9
```

If you see a similar error to "Creating a virtual machine from Marketplace image or a custom image sourced from a Marketplace image requires Plan information in the request." when creating a cluster, you must also set the following flags `--plan-offer`, `--plan-publisher`, `--plan-sku`. For example when creating a cluster with Rocky Linux VMs, add the following flags to your `dkp create cluster azure` command:

- `--plan-offer`, `--plan-publisher` and `--plan-sku`

Individual manifests using the Output Directory flag:

You can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag. This will create multiple files in the specified directory which must already exist:

```
--output-directory=<existing-directory>
```

Refer to the Cluster Creation Customization Choices (see page 1170) section for more information on how to use optional flags such as the `--output-directory` flag.

For more information regarding this flag or others, please refer to the CLI for the dkp create cluster (see page 1853) section of the documentation and select your provider.

Output:

959 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/gcp

```
Generating cluster resources
```

4. Inspect or edit the cluster objects and familiarize yourself with Cluster API before editing the cluster objects as edits can prevent the cluster from deploying successfully. See Azure Customizing CAPI Clusters (see page 1303) .

Familiarize yourself with Cluster API before editing the cluster objects as edits can prevent the cluster from deploying successfully.

For in-depth documentation about the objects, read Concepts[960] in the Cluster API Book.

5. **(Optional)** Modify Control Plane Audit logs - Users can make modifications to the `KubeadmControlplane` cluster-api object to configure different `kubelet` options. See Configuring the Control Plane (see page 1613) if you wish to configure your control plane beyond the existing options that are available from flags.
6. Create the cluster from the objects generated during `dry run` . A warning will appear in the console if the resource already exists and will require you to remove the resource or update your YAML.

   ```
   kubectl create -f ${CLUSTER_NAME}.yaml
   ```

NOTE: If you used the `--output-directory` flag in your `dkp create .. --dry-run` step above, create the cluster from the objects you created by specifying the directory:

```
kubectl create -f <existing-directory>/
```

Output:

Output will be similar to below:

```
cluster.cluster.x-k8s.io/azure-example created
azurecluster.infrastructure.cluster.x-k8s.io/azure-example created
kubeadmcontrolplane.controlplane.cluster.x-k8s.io/azure-example-control-plane created
azuremachinetemplate.infrastructure.cluster.x-k8s.io/azure-example-control-plane
created
secret/azure-example-etcd-encryption-config created
```

960 https://cluster-api.sigs.k8s.io/user/concepts.html

```
machinedeployment.cluster.x-k8s.io/azure-example-md-0 created
azuremachinetemplate.infrastructure.cluster.x-k8s.io/azure-example-md-0 created
kubeadmconfigtemplate.bootstrap.cluster.x-k8s.io/azure-example-md-0 created
clusterresourceset.addons.cluster.x-k8s.io/calico-cni-installation-azure-example
created
configmap/calico-cni-installation-azure-example created
configmap/tigera-operator-azure-example created
clusterresourceset.addons.cluster.x-k8s.io/azure-disk-csi-azure-example created
configmap/azure-disk-csi-azure-example created
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-azure-example created
configmap/cluster-autoscaler-azure-example created
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-azure-example
created
configmap/node-feature-discovery-azure-example created
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-azure-example
created
configmap/nvidia-feature-discovery-azure-example created
```

7. Wait for the cluster control-plane to be ready:

   ```
   kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
   timeout=20m
   ```

   Output:

   ```
   cluster.cluster.x-k8s.io/azure-example condition met
   ```

8. After the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. Konvoy provides a command to describe the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

Output:

```
NAME                                                          READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/azure-example                                         True
3m4s
├─ClusterInfrastructure - AzureCluster/azure-example          True
8m26s
├─ControlPlane - KubeadmControlPlane/azure-example-control-plane  True
3m4s
│ ├─Machine/azure-example-control-plane-l8j9r                 True
3m9s
```

```
| ├─Machine/azure-example-control-plane-slprd                    True
7m17s
| └─Machine/azure-example-control-plane-xhxxg                    True
5m9s
└─Workers
  └─MachineDeployment/azure-example-md-0                         True
4m31s
    ├─Machine/azure-example-md-0-d67567c8b-2674r                 True
5m19s
    ├─Machine/azure-example-md-0-d67567c8b-mbmhk                 True
5m17s
    ├─Machine/azure-example-md-0-d67567c8b-pzg8k                 True
5m17s
    └─Machine/azure-example-md-0-d67567c8b-z8km9                 True
5m17s
```

9. As they progress, the controllers also create Events. List the Events using this command:

   ```
   kubectl get events | grep ${CLUSTER_NAME}
   ```

   For brevity, the example uses `grep`. It is also possible to use separate commands to get Events for specific objects. For example, `kubectl get events --field-selector involvedObject.kind="AzureCluster"` and `kubectl get events --field-selector involvedObject.kind="AzureMachine"`.

Output:

```
15m         Normal    AzureClusterObjectNotFound                 azurecluster
AzureCluster object default/azure-example not found
15m         Normal    AzureManagedControlPlaneObjectNotFound
azuremanagedcontrolplane                              AzureManagedControlPlane object
default/azure-example not found
15m         Normal    AzureClusterObjectNotFound                 azurecluster
AzureCluster.infrastructure.cluster.x-k8s.io "azure-example" not found
8m22s       Normal    SuccessfulSetNodeRef                       machine/azure-
example-control-plane-bmc9b          azure-example-control-plane-fdvnm
10m         Normal    Machine controller dependency not yet met   azuremachine/azure-
example-control-plane-fdvnm     Machine Controller has not yet set OwnerRef
12m         Normal    SuccessfulSetNodeRef                       machine/azure-
example-control-plane-msftd          azure-example-control-plane-z9q45
10m         Normal    SuccessfulSetNodeRef                       machine/azure-
example-control-plane-nrvff          azure-example-control-plane-vmqwx
12m         Normal    Machine controller dependency not yet met   azuremachine/azure-
example-control-plane-vmqwx     Machine Controller has not yet set OwnerRef
14m         Normal    Machine controller dependency not yet met   azuremachine/azure-
example-control-plane-z9q45     Machine Controller has not yet set OwnerRef
14m         Warning   VMIdentityNone
azuremachinetemplate/azure-example-control-plane   You are using Service Principal
authentication for Cloud Provider Azure which is less secure than Managed Identity.
```

```
Your Service Principal credentials will be written to a file on the disk of each VM
in order to be accessible by Cloud Provider. To learn more, see https://
capz.sigs.k8s.io/topics/identities-use-cases.html#azure-host-identity
12m         Warning   ControlPlaneUnhealthy
kubeadmcontrolplane/azure-example-control-plane    Waiting for control plane to pass
preflight checks to continue reconciliation: [machine azure-example-control-plane-
msftd does not have APIServerPodHealthy condition, machine azure-example-control-
plane-msftd does not have ControllerManagerPodHealthy condition, machine azure-
example-control-plane-msftd does not have SchedulerPodHealthy condition, machine
azure-example-control-plane-msftd does not have EtcdPodHealthy condition, machine
azure-example-control-plane-msftd does not have EtcdMemberHealthy condition]
11m         Warning   ControlPlaneUnhealthy
kubeadmcontrolplane/azure-example-control-plane    Waiting for control plane to pass
preflight checks to continue reconciliation: [machine azure-example-control-plane-
nrvff does not have APIServerPodHealthy condition, machine azure-example-control-
plane-nrvff does not have ControllerManagerPodHealthy condition, machine azure-
example-control-plane-nrvff does not have SchedulerPodHealthy condition, machine
azure-example-control-plane-nrvff does not have EtcdPodHealthy condition, machine
azure-example-control-plane-nrvff does not have EtcdMemberHealthy condition]
9m52s       Normal    SuccessfulSetNodeRef                          machine/azure-
example-md-0-84bd8b5f5b-b8cnq        azure-example-md-0-bsc82
9m53s       Normal    SuccessfulSetNodeRef                          machine/azure-
example-md-0-84bd8b5f5b-j8ldg        azure-example-md-0-mjcbn
9m52s       Normal    SuccessfulSetNodeRef                          machine/azure-
example-md-0-84bd8b5f5b-lx89f        azure-example-md-0-pmq8f
10m         Normal    SuccessfulSetNodeRef                          machine/azure-
example-md-0-84bd8b5f5b-pcv7q        azure-example-md-0-vzprf
15m         Normal    SuccessfulCreate                              machineset/azure-
example-md-0-84bd8b5f5b          Created machine "azure-example-md-0-84bd8b5f5b-
j8ldg"
15m         Normal    SuccessfulCreate                              machineset/azure-
example-md-0-84bd8b5f5b          Created machine "azure-example-md-0-84bd8b5f5b-
lx89f"
15m         Normal    SuccessfulCreate                              machineset/azure-
example-md-0-84bd8b5f5b          Created machine "azure-example-md-0-84bd8b5f5b-
pcv7q"
15m         Normal    SuccessfulCreate                              machineset/azure-
example-md-0-84bd8b5f5b          Created machine "azure-example-md-0-84bd8b5f5b-
b8cnq"
15m         Normal    Machine controller dependency not yet met   azuremachine/azure-
example-md-0-bsc82             Machine Controller has not yet set OwnerRef
15m         Normal    Machine controller dependency not yet met   azuremachine/azure-
example-md-0-mjcbn             Machine Controller has not yet set OwnerRef
15m         Normal    Machine controller dependency not yet met   azuremachine/azure-
example-md-0-pmq8f             Machine Controller has not yet set OwnerRef
```

If changing the Calico encapsulation (see page 1095), D2iQ recommends changing it after cluster creation, but before production.

DKP uses Azure CSI as the default storage provider (see page 110). You can use a Kubernetes CSI[961] compatible storage solution that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[962] for more information.
If you're not using the default, you cannot deploy an alternate provider until ***after*** the `dkp create cluster` is finished. However, it must be determined before Kommander installation.

6.14.3.3.1.4 Known Limitations

Be aware of these limitations in the current release of Konvoy.

The Konvoy version used to create a bootstrap cluster must match the Konvoy version used to create a workload cluster.

- Konvoy supports deploying one workload cluster.
- Konvoy generates a set of objects for one Node Pool.
- Konvoy does not validate edits to cluster objects.

6.14.3.3.1.5 Next Step

Azure Make new Cluster Self-Managed (see page 1324)

6.14.3.4 Azure Make new Cluster Self-Managed

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which then deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, which makes the workload cluster self-managed. This section describes how to make a workload cluster self-managed.

This page contains instructions on how to make your cluster self-managed. This is necessary if there is only one cluster in your environment, or if this cluster should become the Management cluster in a multi-cluster environment.

If you already have a self-managed or Management cluster in your environment, you do not need to make this cluster self-managed if you want to attach it instead.

961 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
962 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

6.14.3.4.1 Make the New Kubernetes Cluster a Management Cluster

1. Deploy cluster lifecycle services on the workload cluster:

   ```
   dkp create capi-components --kubeconfig=${CLUSTER_NAME}.conf
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

   The output is similar to:

   ```
   ✓ Initializing new CAPI components
   ```

2. Move the Cluster API objects from the bootstrap to the workload cluster:

   The cluster lifecycle services on the workload cluster are ready, but the workload cluster configuration is on the bootstrap cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the bootstrap to the workload cluster. This process is also called a Pivot[963].

   ```
   dkp move capi-resources --to-kubeconfig=${CLUSTER_NAME}.conf
   ```

   Output:

   ```
   ✓ Moving cluster resources
   You can now view resources in the moved cluster by using the --kubeconfig flag
   with kubectl. For example: kubectl --kubeconfig=azure-example.conf get nodes
   ```

> To ensure only one set of cluster lifecycle services manages the workload cluster, Konvoy first pauses reconciliation of the objects on the bootstrap cluster, then creates the objects on the workload cluster. As Konvoy copies the objects, the cluster lifecycle services on the workload cluster reconcile the objects. The workload cluster becomes self-managed after Konvoy creates all the objects. If it fails, the `move` command can be safely retried.

3. Wait for the cluster control-plane to be ready:

963 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf wait --for=condition=ControlPlaneReady
"clusters/${CLUSTER_NAME}" --timeout=20m
```

Output:

```
cluster.cluster.x-k8s.io/azure-example condition met
```

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

After moving the cluster lifecycle services to the workload cluster, remember to use Konvoy with the workload cluster kubeconfig.

```
dkp describe cluster --kubeconfig=${CLUSTER_NAME}.conf -c ${CLUSTER_NAME}
```

Output:

```
NAME                                                          READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/azure-example                                         True
55s
├─ClusterInfrastructure - AzureCluster/azure-example          True
67s
├─ControlPlane - KubeadmControlPlane/azure-example-control-plane  True
55s
│ ├─Machine/azure-example-control-plane-67f47                 True
58s
│ ├─Machine/azure-example-control-plane-7pllh                 True
65s
│ └─Machine/azure-example-control-plane-jtfgv                 True
65s
└─Workers
  └─MachineDeployment/azure-example-md-0                      True
67s
    ├─Machine/azure-example-md-0-f9cb9c79b-6nsb9              True
59s
    ├─Machine/azure-example-md-0-f9cb9c79b-jxwl6              True
58s
    ├─Machine/azure-example-md-0-f9cb9c79b-ktg7z              True
59s
    └─Machine/azure-example-md-0-f9cb9c79b-nxcm2              True
66s
```

5. Remove the bootstrap cluster, as the workload cluster is now self-managed:

```
dkp delete bootstrap --kubeconfig $HOME/.kube/config
```

Output:

```
✓ Deleting bootstrap cluster
```

6.14.3.4.1.1 Known Limitations

- Konvoy supports moving only one set of cluster objects from the bootstrap cluster to the workload cluster, or vice-versa.
- Konvoy only supports moving all namespaces in the cluster; Konvoy does not support migration of individual namespaces.

6.14.3.4.1.2 Next Step

Azure Explore your New Cluster (see page 1327)

6.14.3.5 Azure Explore your New Cluster

6.14.3.5.1 Learn to interact with your Kubernetes cluster

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster.

Before you start, make sure you have created a workload cluster, as described in Create a New Cluster (see page 1314).

6.14.3.5.2 Explore the New Kubernetes Cluster

1. Get a kubeconfig file for the workload cluster:

   When the workload cluster is created, the cluster lifecycle services generate a kubeconfig file for the workload cluster, and write it to a *Secret*. The kubeconfig file is scoped to the cluster administrator.

   Get the kubeconfig from the *Secret*, and write it to a file, using this command:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

2. List the Nodes using this command:

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
   ```

```
NAME                                 STATUS   ROLES                  AGE
VERSION
azure-example-control-plane-7ffnl    Ready    control-plane,master   6m18s
v1.28.7
azure-example-control-plane-l4bv8    Ready    control-plane,master   14m
v1.28.7
azure-example-control-plane-n4g4l    Ready    control-plane,master   18m
v1.28.7
azure-example-md-0-mpctb             Ready    <none>                 15m     v1.2
8.7
azure-example-md-0-qglp9             Ready    <none>                 15m     v1.2
8.7
azure-example-md-0-sgrd6             Ready    <none>                 16m     v1.2
8.7
azure-example-md-0-wzbkl             Ready    <none>                 16m     v1.2
8.7
```

NOTE: It may take a few minutes for the Status to move to `Ready` while the Pod network is deployed. The Nodes' Status should change to Ready soon after the `calico-node` DaemonSet Pods are Ready.

3. List the Pods using this command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get --all-namespaces pods
```

Output:

```
NAMESPACE                           NAME
READY   STATUS    RESTARTS      AGE
calico-system                       calico-kube-controllers-57fbd7bd59-v4tss
1/1     Running   0             19m
calico-system                       calico-node-59llv
1/1     Running   0             17m
calico-system                       calico-node-7t7wj
1/1     Running   0             16m
calico-system                       calico-node-pf8q8
1/1     Running   0             17m
calico-system                       calico-node-sh2b7
1/1     Running   0             8m17s
calico-system                       calico-node-tmxl5
1/1     Running   0             19m
```

```
calico-system                       calico-node-vt5fh
1/1     Running   0             18m
calico-system                       calico-node-whfs8
1/1     Running   0             18m
calico-system                       calico-typha-797c9666d5-5w99r
1/1     Running   0             19m
calico-system                       calico-typha-797c9666d5-hj6mj
1/1     Running   0             18m
calico-system                       calico-typha-797c9666d5-s7rc6
1/1     Running   0             17m
capa-system                         capa-controller-manager-74fffb5676-ch6xd
1/1     Running   0             11m
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-
manager-867759cc67-vg4lh       1/1     Running   0             15m
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-
manager-5df55579c4-pc8x9   1/1     Running   1 (11m ago)   15m
capi-system                         capi-controller-manager-79cc58bf5f-xsp9t
1/1     Running   0             15m
cappp-system                        cappp-controller-manager-85b5c77497-8ss8r
1/1     Running   0             14m
capv-system                         capv-controller-manager-7bf4d8b66-6x2mx
1/1     Running   0             14m
capz-system                         capz-controller-manager-5d4c6468bf-wfhcc
1/1     Running   0             14m
capz-system                         capz-nmi-2cbrg
1/1     Running   0             14m
capz-system                         capz-nmi-8dllm
1/1     Running   0             14m
capz-system                         capz-nmi-95dfk
1/1     Running   0             14m
capz-system                         capz-nmi-rtnd4
1/1     Running   0             14m
cert-manager                        cert-manager-848f547974-gjc5p
1/1     Running   1 (10m ago)   15m
cert-manager                        cert-manager-cainjector-54f4cc6b5-rnh4f
1/1     Running   0             15m
cert-manager                        cert-manager-webhook-7c9588c76-rn2sd
1/1     Running   0             15m
kube-system                         cluster-autoscaler-68c759fbf6-6vg5r
1/1     Running   1 (11m ago)   20m
kube-system                         coredns-78fcd69978-6gx44
1/1     Running   0             20m
kube-system                         coredns-78fcd69978-gr5q7
1/1     Running   0             20m
kube-system                         csi-azuredisk-controller-c8fb44c8b-jhmfz
6/6     Running   5 (11m ago)   20m
kube-system                         csi-azuredisk-controller-c8fb44c8b-lpbbs
6/6     Running   0             20m
kube-system                         csi-azuredisk-node-2g7vw
3/3     Running   0             8m17s
kube-system                         csi-azuredisk-node-6rdqc
3/3     Running   0             18m
```

```
kube-system                         csi-azuredisk-node-99c6q
3/3     Running   0             17m
kube-system                         csi-azuredisk-node-9b4ms
3/3     Running   0             17m
kube-system                         csi-azuredisk-node-mz5pr
3/3     Running   0             18m
kube-system                         csi-azuredisk-node-r2t99
3/3     Running   0             16m
kube-system                         csi-azuredisk-node-t7gfs
3/3     Running   0             20m
kube-system                         etcd-azure-example-control-plane-7ffnl
1/1     Running   0             8m15s
kube-system                         etcd-azure-example-control-plane-l4bv8
1/1     Running   0             16m
kube-system                         etcd-azure-example-control-plane-n4g4l
1/1     Running   0             19m
kube-system                         kube-apiserver-azure-example-control-plane-7ffnl
1/1     Running   0             8m16s
kube-system                         kube-apiserver-azure-example-control-plane-l4bv8
1/1     Running   0             16m
kube-system                         kube-apiserver-azure-example-control-plane-n4g4l
1/1     Running   0             19m
kube-system                         kube-controller-manager-azure-example-control-
plane-7ffnl         1/1     Running   0             8m17s
kube-system                         kube-controller-manager-azure-example-control-
plane-l4bv8         1/1     Running   0             16m
kube-system                         kube-controller-manager-azure-example-control-
plane-n4g4l         1/1     Running   1 (17m ago)   19m
kube-system                         kube-proxy-82zdl
1/1     Running   0             8m17s
kube-system                         kube-proxy-fd9f9
1/1     Running   0             18m
kube-system                         kube-proxy-l6lgc
1/1     Running   0             17m
kube-system                         kube-proxy-lzswh
1/1     Running   0             16m
kube-system                         kube-proxy-ndfmt
1/1     Running   0             20m
kube-system                         kube-proxy-nxlp9
1/1     Running   0             18m
kube-system                         kube-proxy-v9sxp
1/1     Running   0             17m
kube-system                         kube-scheduler-azure-example-control-plane-7ffnl
1/1     Running   0             8m16s
kube-system                         kube-scheduler-azure-example-control-plane-l4bv8
1/1     Running   0             16m
kube-system                         kube-scheduler-azure-example-control-plane-n4g4l
1/1     Running   1 (17m ago)   19m
node-feature-discovery              node-feature-discovery-master-84c67dcbb6-d2gm7
1/1     Running   0             20m
node-feature-discovery              node-feature-discovery-worker-drgf6
1/1     Running   0             17m
```

```
node-feature-discovery                  node-feature-discovery-worker-hcz6k
1/1     Running   0             17m
node-feature-discovery                  node-feature-discovery-worker-pgbcd
1/1     Running   0             16m
node-feature-discovery                  node-feature-discovery-worker-vhj96
1/1     Running   0             16m
tigera-operator                         tigera-operator-d499f5c8f-jnj8b
1/1     Running   1 (18m ago)   19m
```

6.14.3.5.2.1 Next Step

Azure Install Kommander Non-air-gapped (see page 1331)

6.14.3.6 Azure Install Kommander Non-air-gapped

This section provides installation instructions for the Kommander component of DKP for a non-air-gapped environment in Azure.

6.14.3.6.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

6.14.3.6.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init > kommander.yaml
   ```

4. If required, customize your `kommander.yaml`.

   See Kommander Customizations (see page 1558) for customization options. Some of them include:

   - Custom Domains and Certificates

- HTTP proxy
- Disabling the AI Navigator application
- External Load Balancer
- GPU utilization, etc.
- Rook Ceph customization for Pre-provisioned environments (see page 1541)

5. *If required:* If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
...
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true
...
```

6. Expand **one** of the following sets of instructions, depending on your license and application environments:

Essential License: Install Kommander in a Non-Air-Gapped Environment

6.14.3.6.1.2 Essential License: Install Kommander

Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Enterprise License: Install Kommander in a Non-air-gapped Environment with DKP Catalog Applications

6.14.3.6.1.3 Enterprise License: Install Kommander with DKP Catalog Applications

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.7.0
...
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

6.14.3.6.2 Kommander Customizations

You can configure the Kommander component of DKP during the initial installation, and also post-installation using the DKP CLI. If you are not sure of what you want to customize during install, then proceed to the next step. To read about Kommander component customization options, refer to this section of the documetation: Kommander Customizations (see page 1558)

6.14.3.6.3 Next Step

Day 2 - Cluster Operations Management (see page 590)

6.14.4 Azure Management Tools

After cluster creation and configuration, you can revisit clusters to update and change variables.

- Azure Replace a Node (see page 1334)
- Azure Certificate Renewal (see page 1337)
- Azure Delete Cluster (see page 1339)

6.14.4.1 Azure Replace a Node

6.14.4.1.1 Replace a worker node

6.14.4.1.2 Prerequisites

Before you begin, you must:

- Create a workload cluster (see page 1314).
- Make the new cluster self-managed (see page 1324).

6.14.4.1.3 Replace a worker node

In certain situations, you may want to delete a worker node and have Cluster API[964] replace it with a newly provisioned machine.

1. Identify the name of the node to delete.

   List the nodes:

   ```
   kubectl --kubeconfig ${CLUSTER_NAME}.conf get nodes
   ```

   The output from this command resembles the following:

   ```
   NAME                                STATUS   ROLES                  AGE
   VERSION
   azure-example-control-plane-ckwm4   Ready    control-plane,master   35m
   v1.28.7
   azure-example-control-plane-d4fdf   Ready    control-plane,master   31m
   v1.28.7
   azure-example-control-plane-qrvm9   Ready    control-plane,master   33m
   v1.28.7
   azure-example-md-0-4w7gq            Ready    <none>                 33m   v1.28
   .7
   azure-example-md-0-6gb9k            Ready    <none>                 33m   v1.28
   .7
   azure-example-md-0-p2n8c            Ready    <none>                 11m   v1.28
   .7
   azure-example-md-0-s5zbh            Ready    <none>                 33m   v1.28
   .7
   ```

964 https://cluster-api.sigs.k8s.io/

2. Export a variable with the node name to use in the next steps:

   This example uses the name `azure-example-control-plane-ckwm4`.

   ```
   export NAME_NODE_TO_DELETE="<azure-example-control-plane-ckwm4>"
   ```

3. Delete the Machine resource

   ```
   export NAME_MACHINE_TO_DELETE=$(kubectl --kubeconfig ${CLUSTER_NAME}.conf get
   machine -ojsonpath="{.items[?
   (@.status.nodeRef.name==\"$NAME_NODE_TO_DELETE\")].metadata.name}")
   kubectl --kubeconfig ${CLUSTER_NAME}.conf delete machine
   "$NAME_MACHINE_TO_DELETE"
   ```

   ```
   machine.cluster.x-k8s.io "azure-example-control-plane-slprd" deleted
   ```

   The command will not return immediately. It will return once the Machine resource has been deleted.

   A few minutes after the Machine resource is deleted, the corresponding Node resource is also deleted.

4. Observe that the Machine resource is being replaced using this command:

   ```
   kubectl --kubeconfig ${CLUSTER_NAME}.conf get machinedeployment
   ```

   ```
   NAME                  CLUSTER         REPLICAS   READY   UPDATED   UNAVAILABLE
   PHASE       AGE     VERSION
   azure-example-md-0    azure-example   4          3       4         1
   ScalingUp   7m30s   v1.28.7
   long-running-md-0     long-running    4          4       4         0
   Running     7m28s   v1.28.7
   ```

   In this example, there are two replicas, but only 1 is ready. One replica is unavailable, and the `ScalingUp` phase means a new Machine is being created.

5. Identify the replacement Machine using this command:

```
export NAME_NEW_MACHINE=$(kubectl --kubeconfig ${CLUSTER_NAME}.conf get
machines \
    -l=cluster.x-k8s.io/deployment-name=${CLUSTER_NAME}-md-0 \
    -ojsonpath='{.items[?(@.status.phase=="Running")].metadata.name}{"\n"}')
echo $NAME_NEW_MACHINE
```

```
azure-example-md-0-d67567c8b-2674r azure-example-md-0-d67567c8b-n276j azure-
example-md-0-d67567c8b-pzg8k azure-example-md-0-d67567c8b-z8km9
```

If the output is empty, the new Machine has probably exited the `Provisioning` phase and entered the `Running` phase.

6. Identify the replacement Node using this command:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf get nodes
```

```
NAME                                STATUS   ROLES                  AGE
VERSION
azure-example-control-plane-d4fdf   Ready    control-plane,master   43m
v1.28.7
azure-example-control-plane-qrvm9   Ready    control-plane,master   45m
v1.28.7
azure-example-control-plane-tz56m   Ready    control-plane,master   8m22s
v1.28.7
azure-example-md-0-4w7gq            Ready    <none>                 45m     v1.2
8.7
azure-example-md-0-6gb9k            Ready    <none>                 45m     v1.2
8.7
azure-example-md-0-p2n8c            Ready    <none>                 22m     v1.2
8.7
azure-example-md-0-s5zbh            Ready    <none>                 45m     v1.2
8.7
```

If the output is empty, the Node resource is not yet available, or does not yet have the expected annotation. Wait a few minutes, then repeat the command.

6.14.4.2 Azure Certificate Renewal

6.14.4.2.1 Configure Automated Renewal for Managed Kubernetes PKI Certficates

During cluster creation, Kubernetes establishes a Public Key Infrastructure (PKI) for generating the TLS certificates needed for securing cluster communication for various components such as `etcd`, `kubernetes-apiserver` and `kube-proxy`. The certificates created by these components have a default expiration of one year and are renewed when an administrator updates the cluster.

Kubernetes provides a facility to renew all certificates automatically during control plane updates. For administrators who need long-running clusters or clusters that are not upgraded often, `dkp` provides automated certificate renewal, without a cluster upgrade.

6.14.4.2.2 Requirements

This feature requires Python 3.5 or greater to be installed on all control plane hosts.

6.14.4.2.3 Prerequisites

Prerequisite:

- Complete the bootstrap Cluster Lifecycle (see page 1311) topic.

6.14.4.2.3.1 Create a cluster with automated certificate renewal

To enable the automated certificate renewal, create a Konvoy cluster using the `certificate-renew-interval` flag:

```
dkp create cluster azure --certificate-renew-interval=60 --cluster-name=long-running
```

The `certificate-renew-interval` is the number of days after which Kubernetes-managed PKI certificates will be renewed. For example, a `certificate-renew-interval` value of 60 means the certificates will be renewed every 60 days.

6.14.4.2.3.2 Technical details

The following manifests are modified on the control plane hosts, and are located at `/etc/kubernetes/manifests`. Modifications to these files requires SUDO access.

```
kube-controller-manager.yaml
kube-apiserver.yaml
```

```
kube-scheduler.yaml
kube-proxy.yaml
```

The following annotation indicates the time each component was reset:

```
metadata:
  annotations:
    konvoy.d2iq.io/restartedAt: $(date +%s)
```

This only occurs when the PKI certificates are older than the interval given at cluster creation time. This is activated by a `systemd` `timer` called `renew-certs.timer` that triggers an associated `systemd` service called `renew-certs.service` that runs on all of the control plane hosts.

6.14.4.2.3.3 Debugging

To debug the automatic certificate renewal feature, a cluster administrator can look at several different components to see if the certificates were renewed. For example, an administrator might start with a look at the control plane pod definition to check the last reset time. To determine if a scheduler pod was properly reset, run the command:

```
kubectl get pod -n kube-system kube-scheduler-ip-10-0-xx-xx.us-west-2.compute.interna
l -o yaml
```

The output of the command will be similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    konvoy.d2iq.io/restartedAt: "1626124940.735733"
```

Administrators who want more details on the execution of the `systemd` service can use `ssh` to connect to the control plane hosts, and then use the `systemctl` and `journalctl` commands that follow to help diagnose potential issues.

To check the status of the timers, when they last ran, and when they are scheduled to run next, use the command:

```
systemctl list-timers
```

To check the status of the `renew-certs` service, use the command:

```
systemctl status renew-certs
```

To get the logs of the last run of the service, use the command:

```
journalctl logs -u renew-certs
```

6.14.4.3 Azure Delete Cluster

6.14.4.3.1 Delete the Kubernetes cluster and clean up your environment

6.14.4.3.2 Prepare to Delete a Workload Cluster

> **NOTE:** A self-managed workload cluster cannot delete itself. If your workload cluster is self-managed, you must create a bootstrap cluster and move the cluster lifecycle services to the bootstrap cluster before deleting the workload cluster.

If you did not make your workload cluster self-managed, as described in Make New Cluster Self-Managed (see page 1324), proceed to Delete the workload cluster (see page 1341) section below.

1. Create a bootstrap cluster:

   The bootstrap cluster will host the Cluster API controllers that reconcile the cluster objects marked for deletion:

> **NOTE:** To avoid using the wrong kubeconfig, the following steps use explicit kubeconfig paths and contexts.

```
dkp create bootstrap --kubeconfig $HOME/.kube/config
```

```
✓ Creating a bootstrap cluster
✓ Initializing new CAPI components
```

2. Move the Cluster API objects from the workload to the bootstrap cluster: The cluster lifecycle services on the bootstrap cluster are ready, but the workload cluster configuration is on the workload cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the workload to the bootstrap cluster. This process is also called a Pivot[965].

```
dkp move capi-resources \
    --from-kubeconfig ${CLUSTER_NAME}.conf \
    --from-context ${CLUSTER_NAME}-admin@${CLUSTER_NAME} \
    --to-kubeconfig $HOME/.kube/config \
    --to-context kind-konvoy-capi-bootstrapper
```

```
✓ Moving cluster resources
```

3. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

```
dkp describe cluster --kubeconfig $HOME/.kube/config -c ${CLUSTER_NAME}
```

```
NAME                                                          READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/azure-example                                         True
15s
├─ClusterInfrastructure - AzureCluster/azure-example          True
29s
├─ControlPlane - KubeadmControlPlane/azure-example-control-plane  True
15s
│ ├─Machine/azure-example-control-plane-gvj5d                 True
22s
│ ├─Machine/azure-example-control-plane-l8j9r                 True
23s
│ └─Machine/azure-example-control-plane-xhxxg                 True
23s
└─Workers
  └─MachineDeployment/azure-example-md-0                      True
35s
    ├─Machine/azure-example-md-0-d67567c8b-2674r              True
24s
    ├─Machine/azure-example-md-0-d67567c8b-n276j              True
25s
    ├─Machine/azure-example-md-0-d67567c8b-pzg8k              True
23s
```

965 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
    └─Machine/azure-example-md-0-d67567c8b-z8km9                    True
24s
```

NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use `dkp` with the workload cluster `kubeconfig`.

1. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig $HOME/.kube/config wait --for=condition=controlplaneready
"clusters/${CLUSTER_NAME}" --timeout=60m
```

```
cluster.cluster.x-k8s.io/azure-example condition met
```

Persistent Volumes (PVs) are not deleted automatically by design in order to preserve your data. However, they take up storage space if not deleted. You must delete PVs manually. Information for backup of a cluster and PVs is on the page in documentation called Back up your Cluster's Applications and Persistent Volumes (see page 863).

6.14.4.3.3 Delete the Workload Cluster

1. Make sure your Azure credentials are up to date. Refresh the credentials using this command:

```
dkp update bootstrap credentials azure --kubeconfig $HOME/.kube/config
```

2. To delete a cluster, you would use `dkp delete cluster` and pass in the name of the cluster you are trying to delete with `--cluster-name` flag. You would use `kubectl get clusters` to get those details (`--cluster-name` and `--namespace`) of the Kubernetes cluster to delete it. NOTE: Do not use `dkp get clusters` since that gets you Kommander cluster details rather than Konvoy kubernetes cluster details.

```
kubectl get clusters
```

3. Delete the Kubernetes cluster and wait a few minutes:

   NOTE: Before deleting the cluster, dkp deletes all Services of type LoadBalancer on the cluster. To skip this step, use the flag `--delete-kubernetes-resources=false`.

```
dkp delete cluster --cluster-name=${CLUSTER_NAME} --kubeconfig $HOME/.kube/
config
```

```
✓ Deleting Services with type LoadBalancer for Cluster default/azure-example
✓ Deleting ClusterResourceSets for Cluster default/azure-example
✓ Deleting cluster resources
✓ Waiting for cluster to be fully deleted
Deleted default/azure-example cluster
```

   After the workload cluster is deleted, delete the bootstrap cluster.

6.14.4.3.4 Delete the Bootstrap Cluster

1. Use `dkp` with the bootstrap cluster to delete the workload cluster.

```
dkp delete bootstrap --kubeconfig $HOME/.kube/config
```

```
✓ Deleting bootstrap cluster
```

6.14.4.3.5 Next Step:

Once your cluster is built in the Konvoy component of DKP for your infrastructure/environment, you will install the Kommander (see page 1558) component of DKP to see your dashboard and continue customization.

6.14.4.3.6 Known Limitations

NOTE: Be aware of these limitations in the current release of Konvoy.

- The Konvoy version used to create the workload cluster must match the Konvoy version used to delete the workload cluster.

6.15 AKS Infrastructure

Enterprise Gov Advanced

When installing DKP on Azure Kubernetes Service (**AKS**) infrastructure, you can choose from multiple configuration types.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

The different types of AKS configuration types supported in DKP are covered in this section.

> An AKS cluster cannot be a Management (see page 80) or Essential (see page 81) cluster. To install DKP on your AKS cluster, first ensure you have a Management cluster with DKP and the Kommander component installed, that handles the lifecycle of your AKS cluster.

- Create a New AKS Cluster (see page 1343)
- Explore New AKS Cluster (see page 1349)
- Delete AKS Cluster (see page 1352)

6.15.1 Create a New AKS Cluster

Enterprise Gov Advanced

6.15.1.1 Use DKP to create a new AKS cluster

> Ensure that the `KUBECONFIG` environment variable is set to the Management cluster by running `export KUBECONFIG=<Management_cluster_kubeconfig>.conf`.

6.15.1.2 Name Your Cluster

Give your cluster a unique name suitable for your environment.

> The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[966] for more naming information.

6.15.1.3 Create a New AKS Kubernetes Cluster

1. Set the environment variable to a name for this cluster.

   ```
   export CLUSTER_NAME=<aks-example>
   ```

2. Check to see what version of Kubernetes is available in your region. When deploying with AKS, you need to declare the version of Kubernetes you wish to use by running the following command, substituting `<your-location>` for the Azure region you're deploying to:

   ```
   az aks get-versions -o table --location <your-location>
   ```

3. Set the version of Kubernetes you've chosen:
   NOTE: Refer to the current release Kubernetes compatibility table (see page 139) for correct version to use and choose an available 1.27.x version. The version listed in the command is an example.

   ```
   export KUBERNETES_VERSION=1.27.6
   ```

4. Create the cluster:

   ```
   dkp create cluster aks --cluster-name=${CLUSTER_NAME} --additional-tags=owner=$
   (whoami) --kubernetes-version=${KUBERNETES_VERSION}
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

966 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

```
Generating cluster resources
cluster.cluster.x-k8s.io/aks-example created
azuremanagedcontrolplane.infrastructure.cluster.x-k8s.io/aks-example created
azuremanagedcluster.infrastructure.cluster.x-k8s.io/aks-example created
machinepool.cluster.x-k8s.io/aks-example created
azuremanagedmachinepool.infrastructure.cluster.x-k8s.io/cp6dsz8 created
machinepool.cluster.x-k8s.io/aks-example-md-0 created
azuremanagedmachinepool.infrastructure.cluster.x-k8s.io/mp6gglj created
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-aks-example
created
configmap/cluster-autoscaler-aks-example created
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-aks-example
created
configmap/node-feature-discovery-aks-example created
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-aks-example
created
configmap/nvidia-feature-discovery-aks-example created
```

6.15.1.4 Inspecting or Editing the Cluster Objects

Use your favorite editor.

> **NOTE:** Editing the cluster objects requires some understanding of Cluster API. Edits can prevent the cluster from deploying successfully.

The objects are Custom Resources[967] defined by Cluster API components, and they belong in three different categories:

- Cluster
  A *Cluster* object has references to the infrastructure-specific and control plane objects.
- Control Plane
- Node Pool
  A Node Pool is a collection of machines with identical properties. For example, a cluster might have one Node Pool with large memory capacity, another Node Pool with GPU support. Each Node Pool is described by three objects: The MachinePool references an object that describes the configuration of Kubernetes components (for example, kubelet) deployed on each node pool machine, and an infrastructure-specific object that describes the properties of all node pool machines. Here, it references a *KubeadmConfigTemplate*.

For in-depth documentation about the objects, read Concepts[968] in the Cluster API Book.

1. Wait for the cluster control-plane to be ready:

967 https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/
968 https://cluster-api.sigs.k8s.io/user/concepts.html

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=20m
```

```
cluster.cluster.x-k8s.io/aks-example condition met
```

The `READY` status will become `True` after the cluster control-plane becomes ready.

2. After the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. DKP provides a command to describe the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

```
NAME                                                          READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/aks-example                                           True
48m
├─ClusterInfrastructure - AzureManagedCluster/aks-example
└─ControlPlane - AzureManagedControlPlane/aks-example
```

3. As they progress, the controllers also create Events. List the Events using this command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, the example uses `grep`. It is also possible to use separate commands to get Events for specific objects. For example, `kubectl get events --field-selector involvedObject.kind="AKSCluster"` and `kubectl get events --field-selector involvedObject.kind="AKSMachine"`.

```
48m         Normal    SuccessfulSetNodeRefs               machinepool/aks-
example-md-0                  [{Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000000 UID:e3c30389-660d-46f5-b9d7-219f80b5674d APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000001 UID:300d71a0-f3a7-4c29-9ff1-1995ffb9cfd3 APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000002 UID:8eae2b39-a415-425d-8417-d915a0b2fa52 APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000003 UID:3e860b88-f1a4-44d1-b674-a54fad599a9d APIVersion:
ResourceVersion: FieldPath:}]
```

```
6m4s        Normal    AzureManagedControlPlane available
azuremanagedcontrolplane/aks-example          successfully reconciled
48m         Normal    SuccessfulSetNodeRefs               machinepool/aks-
example                       [{Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000000 UID:e3c30389-660d-46f5-b9d7-219f80b5674d APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000001 UID:300d71a0-f3a7-4c29-9ff1-1995ffb9cfd3 APIVersion:
ResourceVersion: FieldPath:} {Kind: Namespace: Name:aks-mp6gglj-41174201-
vmss000002 UID:8eae2b39-a415-425d-8417-d915a0b2fa52 APIVersion:
ResourceVersion: FieldPath:}]
```

6.15.1.5 Known Limitations

- The DKP version used to create a workload cluster must match the DKP version used to create a workload cluster.
- DKP supports deploying one workload cluster.
- DKP generates a single nodepool is deployed by default, but you can add additional nodepools.
- DKP does not validate edits to cluster objects.

When complete, you can explore the new cluster (see page 1349).

6.15.1.6 Create a new AKS Cluster from the DKP UI

Enterprise Gov Advanced

The DKP user interface allows you to provision an AKS cluster from your browser quickly and easily.

6.15.1.6.1 Prerequisites

6.15.1.6.1.1 Create an AKS Infrastructure Provider

Before provisioning a cluster via the UI, first create an AKS infrastructure provider to hold your AKS credentials:

1. Log in to the Azure command line:

   ```
   az login
   ```

2. Create an Azure Service Principal (SP) by running the following command:

   ```
   az ad sp create-for-rbac --role contributor --name "$(whoami)-konvoy" --
   scopes=/subscriptions/$(az account show --query id -o tsv)
   ```

3. Select **Infrastructure Providers** from the Dashboard menu.
4. Select **Add Infrastructure Provider**.
5. Choose a workspace. (see page 678) If you are already in a workspace, the provider is automatically created in that workspace.
6. Select **Microsoft Azure**.
7. Add a **Name** for your Infrastructure Provider.
8. Take the `id` output from the log in command above and put it into the **Subscription ID** field.
9. Take the `tenant` used in Step 2 and put it into the **Tenant ID** field.
10. Take the `appId` used in Step 2 and put it into the **Client ID** field.
11. Take the `password` used in Step 2 and put it into the **Client Secret** field.
12. Select **Save**.

6.15.1.6.2 Provision an AKS Cluster

Follow these steps to provision an AKS cluster:

1. From the top menu bar, select your target workspace.
2. Select **Clusters > Add Cluster**.
3. Choose **Create Cluster**.
4. Enter the **Cluster Name**.
5. From **Select Infrastructure Provider**, choose the provider created in the prerequisites section.
6. In order to create a **Kubernetes Version**, run the command below in the `az` cli, and then select the version of AKS you want to use.

```
az aks get-versions -o table --location <location>
```

7. Select a data center **location** or specify a custom **location**.
8. Edit your worker **Node Pools**, as necessary. You can choose the **Number of Nodes**, the **Machine Type**, and for the worker nodes, you can choose a **Worker Availability Zone**.
9. Add any additional **Labels** or **Infrastructure Provider Tags**, as necessary.
10. Validate your inputs, then select **Create**.

It can take up to 15 minutes for your cluster to appear in the **Provisioned** status.

You are then redirected to the **Clusters** page, where you'll see your new cluster in the **Provisioning** status. Hover over the status to view the details.

6.15.1.6.3 Access an AKS Cluster

After the cluster is successfully attached (managed), you can retrieve a custom `kubeconfig` file from the UI using your DKP UI administrator credentials.

6.15.2 Explore New AKS Cluster

Enterprise Gov Advanced

6.15.2.1 Learn to interact with your AKS Kubernetes cluster

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster.

Before you start, make sure you have created a workload cluster, as described in Create a New Cluster (see page 1343).

6.15.2.2 Explore the New AKS Cluster

1. Get a kubeconfig file for the workload cluster:

   When the workload cluster is created, the cluster lifecycle services generate a kubeconfig file for the workload cluster, and write it to a *Secret*. The kubeconfig file is scoped to the cluster administrator.

   Get the kubeconfig from the *Secret*, and write it to a file, using this command:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

2. List the Nodes using this command:

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
   ```

   ```
   NAME                                STATUS   ROLES   AGE   VERSION
   aks-cp6dsz8-41174201-vmss000000   Ready    agent   56m   v1.27.11
   aks-cp6dsz8-41174201-vmss000001   Ready    agent   55m   v1.27.11
   aks-cp6dsz8-41174201-vmss000002   Ready    agent   56m   v1.27.11
   aks-mp6gglj-41174201-vmss000000   Ready    agent   55m   v1.27.11
   ```

```
aks-mp6gglj-41174201-vmss000001   Ready    agent   55m   v1.27.11
aks-mp6gglj-41174201-vmss000002   Ready    agent   55m   v1.27.11
aks-mp6gglj-41174201-vmss000003   Ready    agent   56m   v1.27.11
```

NOTE: It may take a few minutes for the Status to move to `Ready` while the Pod network is deployed. The Nodes' Status should change to Ready soon after the `calico-node` DaemonSet Pods are Ready.

3. List the Pods using this command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get --all-namespaces pods
```

```
NAMESPACE                NAME                                       READY
STATUS     RESTARTS   AGE
calico-system            calico-kube-controllers-5dcd4b47b5-tgslm   1/1
Running    0          3m58s
calico-system            calico-node-46dj9                          1/1
Running    0          3m58s
calico-system            calico-node-crdgc                          1/1
Running    0          3m58s
calico-system            calico-node-m7s7x                          1/1
Running    0          3m58s
calico-system            calico-node-qfkqc                          1/1
Running    0          3m57s
calico-system            calico-node-sfqfm                          1/1
Running    0          3m57s
calico-system            calico-node-sn67x                          1/1
Running    0          3m53s
calico-system            calico-node-w2pvt                          1/1
Running    0          3m58s
calico-system            calico-typha-6f7f59969c-5z4t5              1/1
Running    0          3m51s
calico-system            calico-typha-6f7f59969c-ddzqb              1/1
Running    0          3m58s
calico-system            calico-typha-6f7f59969c-rr4lj              1/1
Running    0          3m51s
kube-system              azure-ip-masq-agent-4f4v6                  1/1
Running    0          4m11s
kube-system              azure-ip-masq-agent-5xfh2                  1/1
Running    0          4m11s
```

```
kube-system              azure-ip-masq-agent-9hlk8                        1/1
Running    0          4m8s
kube-system              azure-ip-masq-agent-9vsgg                        1/1
Running    0          4m16s
kube-system              azure-ip-masq-agent-b9wjj                        1/1
Running    0          3m57s
kube-system              azure-ip-masq-agent-kpjtl                        1/1
Running    0          3m53s
kube-system              azure-ip-masq-agent-vr7hd                        1/1
Running    0          3m57s
kube-system              cluster-autoscaler-b4789f4bf-qkfk2               0/1
Init:0/1   0          3m28s
kube-system              coredns-845757d86-9jf8b                          1/1
Running    0          5m29s
kube-system              coredns-845757d86-h4xfs                          1/1
Running    0          4m
kube-system              coredns-autoscaler-5f85dc856b-xjb5z              1/1
Running    0          5m23s
kube-system              csi-azuredisk-node-4n4fx                         3/3
Running    0          3m53s
kube-system              csi-azuredisk-node-8pnjj                         3/3
Running    0          3m57s
kube-system              csi-azuredisk-node-sbt6r                         3/3
Running    0          3m57s
kube-system              csi-azuredisk-node-v25wc                         3/3
Running    0          4m16s
kube-system              csi-azuredisk-node-vfbxg                         3/3
Running    0          4m11s
kube-system              csi-azuredisk-node-w5ff5                         3/3
Running    0          4m11s
kube-system              csi-azuredisk-node-zzgqx                         3/3
Running    0          4m8s
kube-system              csi-azurefile-node-2rpcc                         3/3
Running    0          3m57s
kube-system              csi-azurefile-node-4gqkf                         3/3
Running    0          4m11s
kube-system              csi-azurefile-node-f6k8m                         3/3
Running    0          4m16s
kube-system              csi-azurefile-node-k72xq                         3/3
Running    0          4m8s
kube-system              csi-azurefile-node-vx7r4                         3/3
Running    0          3m53s
kube-system              csi-azurefile-node-zc8kr                         3/3
Running    0          4m11s
kube-system              csi-azurefile-node-zkl6b                         3/3
Running    0          3m57s
kube-system              kube-proxy-4fpb6                                 1/1
Running    0          3m53s
kube-system              kube-proxy-6qfbf                                 1/1
Running    0          4m16s
kube-system              kube-proxy-6wnt2                                 1/1
Running    0          4m8s
```

```
kube-system                 kube-proxy-cspd5                                  1/1
Running     0          3m57s
kube-system                 kube-proxy-nsgq6                                  1/1
Running     0          4m11s
kube-system                 kube-proxy-qz2st                                  1/1
Running     0          4m11s
kube-system                 kube-proxy-zvh9k                                  1/1
Running     0          3m57s
kube-system                 metrics-server-6bc97b47f7-ltkkj                   1/1
Running     0          5m28s
kube-system                 tunnelfront-77d68f78bf-t78ck                      1/1
Running     0          5m23s
node-feature-discovery      node-feature-discovery-master-65dc499cd-fxwb5     1/1
Running     0          3m28s
node-feature-discovery      node-feature-discovery-worker-277xc               1/1
Running     0          3m28s
node-feature-discovery      node-feature-discovery-worker-4dq5k               1/1
Running     0          3m28s
node-feature-discovery      node-feature-discovery-worker-57nb8               1/1
Running     0          3m28s
node-feature-discovery      node-feature-discovery-worker-b4lkl               1/1
Running     0          3m28s
node-feature-discovery      node-feature-discovery-worker-kslst               1/1
Running     0          3m28s
node-feature-discovery      node-feature-discovery-worker-ppjtm               1/1
Running     0          3m28s
node-feature-discovery      node-feature-discovery-worker-x5bgf               1/1
Running     0          3m28s
tigera-operator             tigera-operator-74c4d9cf84-k7css                  1/1
Running     0          5m25s
```

When ready, you can delete the cluster (see page 1352).

6.15.3 Delete AKS Cluster

Enterprise Gov Advanced

Ensure that the `KUBECONFIG` environment variable is set to the self-managed cluster by running `export KUBECONFIG={SELF_MANAGED_AZURE_CLUSTER}.conf`

6.15.3.1 Delete the AKS cluster and clean up your environment

6.15.3.2 Delete the Workload Cluster

1. Delete the Kubernetes cluster and wait a few minutes:

   Before deleting the cluster, DKP deletes all Services of type LoadBalancer on the cluster. Deleting the Service deletes the Azure LoadBalancer that backs it. To skip this step, use the flag `--delete-kubernetes-resources=false`.

> **NOTE:** Do not skip this step if the Azure Network is managed by DKP. When DKP deletes cluster, it deletes the Network.

```
dkp delete cluster --cluster-name=${CLUSTER_NAME}
```

```
 ✓ Deleting Services with type LoadBalancer for Cluster default/aks-example
 ✓ Deleting ClusterResourceSets for Cluster default/aks-example
 ✓ Deleting cluster resources
 ✓ Waiting for cluster to be fully deleted
Deleted default/aks-example cluster
```

6.15.3.3 Next Step:

After your cluster is built in the Konvoy component of DKP for your infrastructure/environment, you will Install the Kommander (see page 1558) component of DKP to see your dashboard and continue customization.

6.15.3.4 Known Limitations

> **NOTE:** Be aware of these limitations in the current release of DKP.

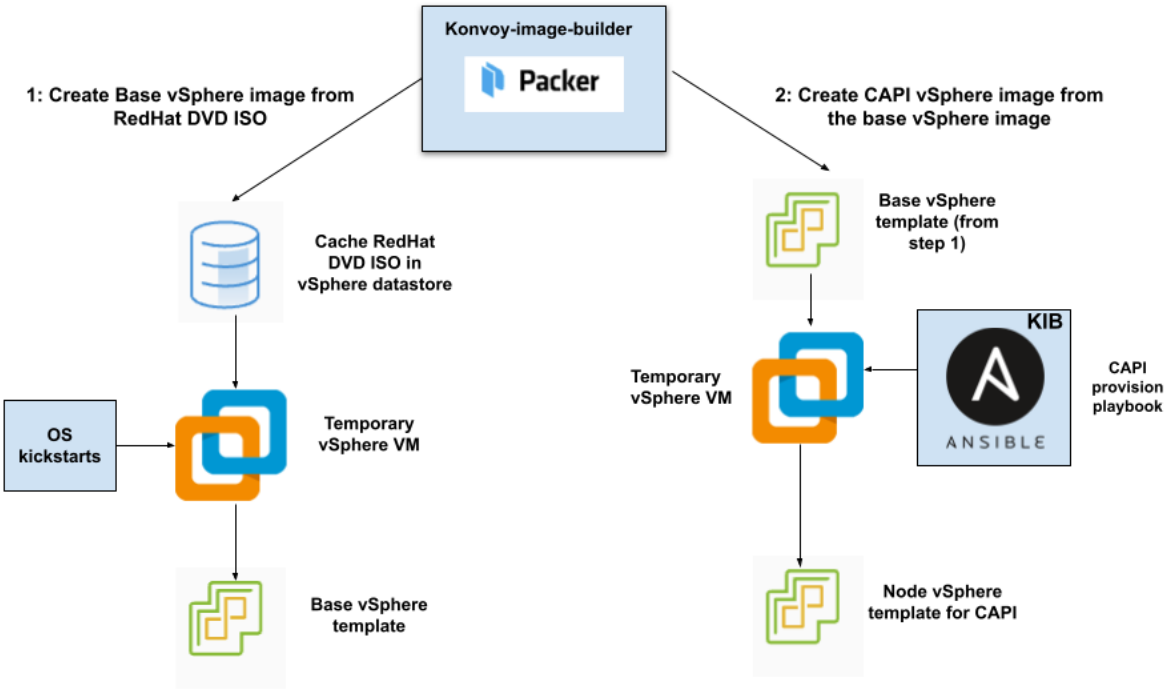
- The DKP version used to create the workload cluster must match the DKP version used to delete the workload cluster.

6.16 vSphere Infrastructure

The overall process for configuring vSphere and DKP together includes the following steps for creating DKP clusters in a VMware vSphere environment:

1. Configure vSphere to provide the needed elements, described in the Prerequisites (see page 1356).
2. Create a bastion VM host (see page 1068) if you are using an air-gapped environment.
3. Create a base OS image (for use in the OVA package containing the disk images packaged with the OVF).
4. Create a CAPI VM image template that uses the base OS image from vCenter and adds the needed Kubernetes cluster components.
5. Create a bootstrap cluster.
6. Create a new DKP cluster on vSphere.
7. Make the cluster self-managing.
8. Explore the cluster and perform other functions as needed.

This diagram illustrates the image creation process:



The workflow on the left shows the creation of a base OS image in the vCenter vSphere client using inputs from Packer. The workflow on the right shows how DKP uses that same base OS image to create CAPI-enabled VM images for your cluster.

After creating the base image, the Konvoy Image Builder (see page 1626) uses it to create a CAPI-enabled vSphere template that includes the Kubernetes objects for the cluster. You can use that resulting template with the DKP `create cluster` command to create the VM nodes in your cluster directly on a vCenter server. From that point, you can use DKP to provision and manage your cluster.

DKP communicates with the code in vCenter Server[969] as the management layer for creating and managing virtual machines after ESXi 6.7 Update 3 or later[970] is installed and configured.

6.16.1 Next Step

To get started, fulfill the vSphere Prerequisites (see page 1356) on the following page.

969 https://docs.vmware.com/en/VMware-vSphere/index.html
970 https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.esxi.install.doc/GUID-B2F01BF5-078A-4C7E-B505-5DFFED0B8C38.html

6.16.2 vSphere Prerequisites

6.16.2.1 Prepare your environment to run DKP with VMware vSphere

Fulfilling the prerequisites involves completing these two areas:

1. DKP prerequisites
2. vSphere prerequisites - vCenter Server + ESXi

6.16.2.2 1. DKP Prerequisites

Before using DKP to create a vSphere cluster, verify that you have:

- An x86_64-based Linux® or macOS® machine.
- Download DKP binaries and Konvoy Image Builder (KIB) (see page 76) image bundle for Linux or macOS.
- A Container engine/runtime installed is required to install DKP and bootstrap:
  - Version Docker®[971] container engine version 18.09.2 or higher installed for Linux or MacOS.
  - Version 4.0 of Podman[972] or higher for Linux. Host requirements found here: Host Requirements[973]
- A registry needs installed on the host where the DKP Konvoy CLI runs. For example, if you are installing Konvoy on your laptop, ensure the laptop has a supported version of Docker or other registry.

On macOS, Docker runs in a virtual machine. Configure this virtual machine with at least 8GB of memory.

- kubectl[974] 1.21.6 for interacting with the running cluster, installed on the host where the DKP Konvoy command line interface (CLI) runs.
- A valid VMware vSphere account with credentials configured.

DKP uses the vsphere CSI driver as the default storage provider (see page 110). Use a Kubernetes CSI[975] compatible storage that is suitable for production.

971 https://docs.docker.com/get-docker/
972 https://podman.io/getting-started/installation
973 https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
974 https://kubernetes.io/docs/tasks/tools/#kubectl
975 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

You can choose from any of the storage options[976] available for Kubernetes. To disable the default that Konvoy deploys, set the default StorageClass as non-default. Then set your newly created StorageClass to be the default by following the commands in the Kubernetes documentation called Changing the Default Storage Class[977].

6.16.2.3 2. VMware vSphere Prerequisites

Before installing, verify that your VMware vSphere Client environment[978] meets the following basic requirements:

- Access to a bastion VM, or other network connected host, running vSphere Client version v6.7.x with Update 3 or later version.
    - You must be able to reach the vSphere API endpoint from where the Konvoy command line interface (CLI) runs.
- vSphere account with credentials configured - this account must have Administrator privileges.
- A RedHat® subscription with user name and password for downloading DVD ISOs.
- For air-gapped environments, a bastion VM host template (see page 127) with access to a configured local registry (see page 1606). VMWare site has more information on vSphere Bastion Hosts[979]. The recommended template naming pattern is `../folder-name/dkp-e2e-bastion-template` or similar. Each infrastructure provider has its own set of bastion host instructions. Refer to your provider's site for details - Azure[980], AWS[981], GCP[982], or vSphere[983].
- Valid vSphere values for the following:
    - vCenter API server URL.
    - Datacenter name.
    - Zone name that contains ESXi hosts[984] for your cluster's nodes.
    - Datastore name for the shared storage resource to be used for the VMs in the cluster.
        - Use of PersistentVolumes in your cluster depends on Cloud Native Storage (CNS), available in vSphere v6.7.x with Update 3 and later versions. CNS depends on this shared Datastore's configuration.
    - Datastore URL from the datastore record for the shared datastore you want your cluster to use.

976 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
977 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/
978 https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.vsphere.vm_admin.doc/GUID-55238059-912E-411F-A0E9-A7A536972A91.html
979 https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.security.doc/GUID-6975426F-56D0-4FE2-8A58-580B40D2F667.html
980 https://learn.microsoft.com/en-us/azure/bastion/quickstart-host-portal
981 https://aws.amazon.com/solutions/implementations/linux-bastion/
982 https://blogs.vmware.com/cloud/2021/06/02/intro-google-cloud-vmware-engine-bastion-host-access-iap/
983 https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.security.doc/GUID-6975426F-56D0-4FE2-8A58-580B40D2F667.html
984 https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.esxi.install.doc/GUID-B2F01BF5-078A-4C7E-B505-5DFFED0B8C38.html

        - You need this URL value to ensure that the correct Datastore is used when DKP creates VMs for your cluster in vSphere.
    - Folder name.
    - Base template name, such as base-rhel-8, or base-rhel-7.
    - Name of a Virtual Network that has DHCP enabled for both air-gapped and non-air-gapped environments.
    - Resource Pools - at least one resource pool needed, with every host in the pool having access to shared storage, such as VSAN.
        - Each host in the resource pool needs access to shared storage, such as NFS or VSAN, to make use of MachineDeployments and high-availability control planes.

6.16.2.3.1 Next Step

vSphere Roles (see page 1358)

6.16.2.4 vSphere Roles

When provisioning Kubernetes clusters with the DKP vSphere provider, there are four roles needed for DKP to provide proper permissions. Roles in vSphere are more like a policy statement for the objects in a vSphere inventory. The Role is assigned to a user and the Object assignment can be inherited by any siblings if desired through propagation.

Add the permission at the highest level and set to propagate the permissions. In small vSphere environments, with just a few hosts, assigning the role/user at the top level and propagating to child resources could be appropriate. However, in the majority of cases this is not possible since security teams will enforce strict restrictions of who should have access to specific resources.

In the table below we describe the level at which these permissions should be assigned.

Level	**Required**	**Propagate to Child**
vCenter Server (Top Level)	No	No
Data Center	Yes	No
Resource Pool	Yes	No
Folder	Yes	Yes
Template	Yes	No

6.16.2.4.1 vCenter Steps to Add Roles

When a user needs permissions less than Admin, a role must be created. The process for configuring a vSphere role includes the following steps in vCenter:

1. Open a vSphere Client connection to the vCenter Server, described in the Prerequisites (see page 1356).
2. Select Home > Administration > Roles > Add Role.
3. Give the new Role a name from the four choices detailed in the next section.
4. Select the Privileges from the permissions directory tree drop-down below each of the four roles.

The list of permissions can be set so that the provider is able to create/modify/delete resources or clone templates, VMs, disks, attach network, etc.

6.16.2.4.2 Create the Four Necessary Roles

The following four Roles should be created for proper DKP access to the required Resource(s) on the correct level of vCenter and resource pools. Set your roles to contain the permissions shown in the permissions directory tree under each role.

1. `dkp-vcenter` - This root level permissions role applies to the Resource:
    a. `vcenter root`

Permissions Tree for dkp-vcenter Role

Resource		
	☑	View
Cns		
	☑	Searchable
Profile-driven storage		
	☑	Profile-driven storage view
Network		
	Session	

	☑	ValidateSession

2. `dkp-datacenter` - This datacenter role allows datacenter, cluster and host related view permissions that need to be assigned to each of them. It applies to the Resources:
    a. `datacenter`
    b. `cluster`
    c. `esx host 1`
    d. `esx host 2`

Do not propagate them because it would give the user *view* privileges on all folders and resource pools.

Permissions Tree for dkp-datacenter Role

??

Resource		
	☑	View
Data Center		
	☑	View
Cluster		
	☑	View
ESX Host 1		
	☑	View
ESX Host 2		
	☑	View

3. `dkp-k8srole` - This role allows CAPV to create resources and assign networks. It is the most extensive permissions, but is only assigned on folder, resource pool, data store and network level so it can easily be separated from other environments. It applies to the Resources:
    a. `resource pool`
    b. `dkp folder`
    c. `dkp data store`
    d. `network`

This role can be propagated to other Resources if desired.

Permissions Tree for dkp-k8srole Role

Permissions Tree

Resource		
	☑	View
Datastore		
	☑	Allocate space
	☑	Browse
	☑	Delete File
	☑	File Management
	☑	Update Virtual Machine File
	☑	Update Virtual Machine Data
Global		
	☑	Set Custom Field

Network		
	☑	Assign network
Resource		
	☑	Assign vApp to Pool
	☑	Assign VM to Pool
Scheduled Task		
	☑	Create
	☑	Delete
	☑	Edit
	☑	Run
	Session	
	☑	ValidateSession
Storage Profile		
	☑	View
Storage Views		
	☑	View

4. `dkp-readonly` -This *optional* role allows the role to clone from templates in other folders and data stores, but not have write access. It applies to the Resources:
    a. `templates folder`
    b. `templates data store`

Permissions Tree for dkp-readonly Role

Datastore		
	☑	View
Folder		
	☑	View
vApp		
	☑	Clone
	☑	Export
Provisioning		
	☑	Clone
	☑	Clone template
	☑	Deploy template

6.16.2.4.2.1 Next Step

Create a Base OS image in vSphere vCenter (see page 1366)

6.16.2.4.3 vSphere Minimum User Permissions

When a user needs permissions less than Admin, a role must be created with those permissions.

In small vSphere environments, with just a few hosts, assigning the role/user at the top level and propagating to child resources could be appropriate as shown on this page in the permissions tree below.

However, in the majority of cases this is not possible as security teams will enforce strict restrictions of who should have access to specific resources.

The process for configuring a vSphere role with the permissions for provisioning nodes and installing includes the following steps:

1. Open a vSphere Client connection to the vCenter Server, described in the Prerequisites (see page 1356).
2. Select Home > Administration > Roles > Add Role.
3. Give the new role a name, then select these Privileges:

Cns		
	☑	Searchable
Datastore		
	☑	Allocate space
	☑	Low level file operations
Host		
	• Configuration	
	☑	Storage partition configuration
Profile-driven storage		
	☑	Profile-driven storage view
Network		
	☑	Assign network
Resource		
	☑	Assign virtual machine to resource pool
Virtual machine		
	• Change Configuration - from the list in that section, select these permissions below:	
	☑	Add new disk

	☑	Add existing disk
	☑	Add or remove device
	☑	Advanced configuration
	☑	Change CPU count
	☑	Change Memory
	☑	Change Settings
	☑	Reload from path
	Edit inventory	
	☑	Create from existing
	☑	Remove
	Interaction	
	☑	Power off
	☑	Power on
	Provisioning	
	☑	Clone template
	☑	Deploy template
	Session	
	☑	ValidateSession

In the table below we describe the level at which these permissions should get assigned to.

Level	Required	Propagate to Child
vCenter Server (Top Level)	No	No
Data Center	Yes	No
Resource Pool	Yes	No
Folder	Yes	Yes
Template	Yes	No

6.16.2.4.3.1 Next Step

Create a Base OS image in vSphere vCenter (see page 1366)

6.16.2.5 Create a Base OS image in vSphere vCenter

Creating a base OS image from DVD ISO files is a one-time process. Building a base OS image creates a base vSphere template in your vSphere environment. The base OS image is used by Konvoy Image Builder (see page 1626)(KIB) to create a VM template to configure Kubernetes nodes by the DKP vSphere provider. For more information about images in vSphere vCenter, refer to VMware documentation[985].

6.16.2.5.1 Create the Base OS Image

For vSphere, a username and password is populated by `SSH_USERNAME` and the user can use authorization via `SSH_PASSWORD` or `SSH_PRIVATE_KEY_FILE` environment variables and required by default by packer[986]. This user should have administrator privileges. It is possible to configure a custom user and password when building the OS image, however, that requires the Konvoy Image Builder (KIB) configuration to be overridden (see page 1670).

While creating the base OS image, it is important to take into consideration the following elements:

- Storage configuration: D2iQ recommends customizing disk partitions and not configuring a SWAP partition.
- Network configuration: as KIB must download and install packages, activating the network is required.
- Connect to Red Hat: if using RHEL, registering with Red Hat is required to configure software repositories and install software packages.

985 https://docs.vmware.com/en/VMware-Cloud-Foundation/4.5/vcf-deploy/GUID-2674DA5A-8DF7-4212-A4A9-88CD798DC303.html

986 https://github.com/mesosphere/konvoy-image-builder/blob/0523fd2c5e6e1ad1d4962f60a47039aa145a6e42/pkg/packer/manifests/vsphere/packer.pkr.hcl

- Software selection: D2iQ recommends choosing **Minimal Install**.
- DKP recommends to install with the packages provided by the operating system package managers. Use the version that corresponds to the major version of your operating system.

6.16.2.5.1.1 Disk Size

For each cluster you create using this base OS image, ensure you establish the disk size of the **root file system** based on:

- The minimum DKP Resource Requirements (see page 119).
- The minimum storage requirements for your organization.

Defaults

Clusters are created with a default disk size of 80 GB.

> ⚠ **For clusters created with the default disk size, the base OS image root file system must be exactly 80 GB.** The root file system cannot be reduced automatically when a machine first boots.

Customization

You can also specify a custom disk size when you create a cluster (see page 0) (see the flags available for use with the vSphere Create Cluster (see page 1861) command). This allows you to use one base OS image to create multiple clusters that have different storage requirements.

> ⚠ Before specifying a disk size when you create a cluster, take into account:
>
> 1. **For some base OS images, the custom disk size option has no effect on the size of the root file system.** This is because some root file systems, for example, those contained in an LVM Logical Volume, cannot be resized automatically when a machine first boots.
> 2. **The specified custom disk size must be equal to, or larger than the size of the base OS image root file system.** This is because a root file system cannot be reduced automatically when a machine first boots.
> 3. In VMware Cloud Director (see page 1444), the base image determines the minimum storage available for the VM.

This Base OS Image is later used to vSphere Create a VM Template (see page 1379) during installation and cluster creation.

If using Flatcar, the documentation from Flatcar regarding disabling or enabling autologin in the Base OS Image is found here: In a vSphere or Pre-provisioned environment, anyone with access to the console of a Vitrual Machine(VM) has access to the core operating system user. This is called autologin. To disable

autologin, you add paramaters to your base Flatcar image. The ways to do this are described in the Flatcar documentation: https://www.flatcar.org/docs/latest/installing/cloud/vmware/#disablingenabling-autologin and https://www.flatcar.org/docs/latest/setup/customization/other-settings/#adding-custom-kernel-boot-options .

6.16.2.5.1.2 Next Step

vSphere Storage Options (see page 1368)

6.16.2.6 vSphere Storage Options

6.16.2.6.1 Explore storage options and considerations for using DKP with VMware vSphere

The vSphere Container Storage[987] plugin supports shared NFS, vNFS, and vSAN. You need to provision your storage options in vCenter prior to creating a CAPI image (see page 1404) in DKP for use with vSphere.

DKP has integrated the CSI 2.x driver used in vSphere. When creating your DKP cluster, DKP uses whatever configuration you provide for the Datastore name. vSAN is not required. Using NFS can reduce the amount of tagging and permission granting required to configure your cluster.

6.16.2.6.2 Next Step

vSphere Cluster Creation Customization Choices (see page 1368)

6.16.3 vSphere Cluster Creation Customization Choices

Below are a some methods to customize your cluster during creation. If none of these choices apply, skip this page and proceed to either section:

- Install vSphere in a Non-air-gapped Environment (see page 1379)
- Install vSphere in an Air-Gapped Environment (see page 1404)

6.16.3.1 Section Topics

When creating clusters, many options are available such as those listed in this section of the documentation. Familiarize yourself with the flags required to apply these customizations during cluster creation.

- vSphere Customizing CAPI Clusters (see page 1369) — Familiarize yourself with Cluster API before editing the cluster objects because edits can prevent the cluster from deploying successfully.
- vSphere Registry Mirrors (see page 1370) — Configure your cluster to use an existing local registry when attempting to pull images by adding the flag(s) to the dkp create cluster command to pull images from your local registry.

987 https://docs.vmware.com/en/VMware-vSphere-Container-Storage-Plug-in/2.0/vmware-vsphere-csp-getting-started/GUID-74AF02D7-1562-48BD-A9FE-C81A53342AC3.html

- vSphere Load the Registry (see page 1371) — Because air-gapped environments do not have direct access to the Internet, you must download, extract and load several required images to your local container registry, before installing DKP.
- vSphere HTTP Proxy (see page 1373) — When creating a DKP cluster in environments that use an HTTP/HTTPS proxy, you must provide proxy details. The proxy values are strings that list a set of proxy servers, URLs, or wildcard addresses that is specific to your environment.
- vSphere Provision on the Flatcar Linux OS (see page 1375) — When provisioning onto the Flatcar Container Linux distribution, you must instruct the bootstrap cluster to make some changes related to the installation paths. To accomplish this, add the --os-hint flatcar flag to the dkp create cluster command.
- Configure MetalLB for a vSphere infrastructure (see page 1376) — It is recommended that an external load balancer(LB) be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests. If you do not have one, you can use Metal LB to create a MetalLB configmap for your vSphere infrastructure.
- vSphere Output Directory YAML (see page 1378) — You can create individual files with different smaller manifests for ease in editing using the --output-directory flag used with --output=json|yaml. You create the directory of where to output resources to files.

6.16.3.1.1 Related Topics

- DKP Concepts and Terms (see page 78)
- Using KIB with vSphere (see page 1652)

6.16.3.1.2 Next Step

After you make decisions about customization choices, proceed to either the non-air-gapped or air-gapped environment instructions:

- AWS Install in a Non-air-gapped Environment (see page 1184)
- AWS Install in an Air-gapped Environment (see page 1207)

6.16.3.2 vSphere Customizing CAPI Clusters

Familiarize yourself with Cluster API (see page 1061) before editing the cluster objects because edits can prevent the cluster from deploying successfully.

The result of this command will allow such edits:

```
dkp create cluster vsphere \
    --cluster-name=${CLUSTER_NAME} \
    --dry-run \
    --output=yaml \
  > ${CLUSTER_NAME}.yaml
```

To edit the YAML, you need to understand the CAPI components to avoid breaking the cluster. For expanded component explanation, refer to the Universal Configurations for All Providers section of the documentation in the section: Customizing CAPI Components for a Cluster (see page 1061)

6.16.3.2.1 Next Topic

vSphere Registry Mirrors (see page 1370)

6.16.3.3 vSphere Registry Mirrors

Configure your cluster to use an existing local registry (see page 1606) when attempting to pull images by adding the flag(s) to the `dkp create cluster` command to pull images from your local registry.

Kubernetes does not natively provide a registry for hosting the container images you will use to run the applications you want to deploy on Kubernetes. Instead, Kubernetes requires you to use an external solution for storing and sharing container images. There are a variety of Kubernetes-compatible registry options that are compatible with DKP.

6.16.3.3.1 How Does it Work?

The **first time** you request an image from your local registry mirror, it pulls the image from the public registry (such as Docker) and stores it locally before handing it back to you. On subsequent requests, the local registry mirror is able to serve the image from its own storage.

6.16.3.3.2 Air-gapped vs Non-air-gapped Environments

In a non-air-gapped environment, you have access to the Internet. You retrieve artifacts from specialized repositories dedicated to them, such as Docker images contained in DockerHub and Helm Charts that come from a dedicated Helm Chart repository. You can also create your own local repository to hold the downloaded container images needed or any custom images you've created with the Konvoy Image Builder (see page 1626) tool.

In an **air-gapped environment**, you need a local repository to store Helm charts, Docker images, and other artifacts. Private registries provide security and privacy into enterprise container image storage, whether hosted remotely or on-premises locally in an air-gapped environment. DKP in an air-gapped environment **requires** a local container registry of trusted images to enable production-level Kubernetes cluster management. However, a local registry is an option in a non-air-gapped environment as well for speed and security.

If you want to use images from this local registry to deploy applications inside your Kubernetes cluster, you'll need to set up a **secret** for a private registry. The secret contains your login data, which Kubernetes needs to connect to your private repository.

6.16.3.3.2.1 Related Topics

More information and detail can be found:

- Registry Mirror Tools (see page 1606)
- Use a Registry Mirror (see page 1609)

6.16.3.3.2.2 Next Step

vSphere Load the Registry (see page 1371)

6.16.3.4 vSphere Load the Registry

Because air-gapped environments do not have direct access to the Internet, you must download, extract and load several required images to your local container registry, before installing DKP.

If desired, environments that are non-air-gapped can also perform the follow steps to use a local registry for speed and security reasons.

6.16.3.4.1 Load Images into your Registry

This page assumes you have already downloaded (see page 76) and extracted (see page 1211) the bundle from the Prerequisites (see page 1151). The sections below explain further how you are pushing the images to your registry and then using them in creating a cluster.

6.16.3.4.2 Download all Images for Air-gapped Deployments

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for prerequisites to download and then how to push the necessary images to this registry.

1. Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`) to load registry images as explained below.
2. Connectivity with clusters attaching to the management cluster is required:
    - Both management and attached clusters must be able to connect to the local registry.
    - The management cluster must be able to connect to all attached cluster's API servers.
    - The management cluster must be able to connect to any load balancers created for platform services on the management cluster.

6.16.3.4.3 Extract Air-gapped Images and Set Variables

Follow these steps to **extract** the air-gapped image bundles into your private registry:

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`, extract the tarball to a local directory:

    ```
    tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
    ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

```
cd dkp-v2.8.1
```

3. Set an environment variable with your registry address:

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

- `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- NOTE: Other local registries may use the options below:
    - JFrog - `CONTAINER_REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
    - `CONTAINER_REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
    - `CONTAINER_REGISTRY_PASSWORD` : optional if username is not set.

6.16.3.4.3.1 Load Images to your Private Registry - Konvoy

Before creating or upgrading a Kubernetes cluster, you need to load the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, refer to Local Registry Tools (see page 1606) page for more information.

Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

6.16.3.4.3.2 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

6.16.3.4.3.3 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME}
--to-registry-password=${REGISTRY_PASSWORD}
```

For more information to set up a private registry with a registry mirror, see this page (see page 1609) for details on using that flag.

6.16.3.4.3.4 Next Step

vSphere HTTP Proxy (see page 1373)

6.16.3.5 vSphere HTTP Proxy

When creating a DKP cluster in environments that use an HTTP/HTTPS proxy, you must provide proxy details. The proxy values are strings that list a set of proxy servers, URLs, or wildcard addresses that is specific to your environment.

If your environment uses HTTP/HTTPS proxies, you must include the flags and their related values in commands for the proxy to be successful throughout various steps of installation:

- `--http-proxy`
- `--https-proxy`
- `--no-proxy`

Create the bootstrap cluster and CAPI components using the appropriate commands, `dkp create bootstrap` and `dkp create capi-components` respectively, combined with the command line flags to include your HTTP/S proxy information.

You can also specify HTTP/S proxy information in an override file when using Konvoy Image Builder (KIB) (see page 1626).

Without these values provided as part of the relevant `dkp create` command, DKP cannot create the requisite parts of your new cluster correctly. This is true of both management and managed clusters (see page 79) alike.

To create a proxied environment, you need to include flags at these action item points:

- Bootstrap cluster
- CAPI components
- Cluster creation
- DKP Kommander component

For full HTTP Proxy configuration, you need to specify proxy settings using all the details in the Configuring an HTTP/HTTPS Proxy (see page 1053)section of the documentation for:

- Configure the Bootstrap Cluster HTTP Proxy Settings (see page 1054)
- Create CAPI Components with HTTP/HTTPS Proxy Settings (see page 1055)
- Create a Cluster with HTTP/HTTPS Proxy (see page 1056)
- Configure an HTTP/HTTPS Proxy for the DKP Kommander Component (see page 1058)

6.16.3.5.1 HTTP Proxy Cluster Example

```
dkp create cluster vsphere \
 --cluster-name ${CLUSTER_NAME} \
 --control-plane-http-proxy="${CONTROL_PLANE_HTTP_PROXY}" \
 --control-plane-https-proxy="${CONTROL_PLANE_HTTPS_PROXY}" \
 --control-plane-no-proxy="${CONTROL_PLANE_NO_PROXY}" \
 --worker-http-proxy="${WORKER_HTTP_PROXY}" \
 --worker-https-proxy="${WORKER_HTTPS_PROXY}" \
 --worker-no-proxy="${WORKER_NO_PROXY}"
```

6.16.3.5.2 Next Topic

vSphere Provision on the Flatcar Linux OS (see page 1375)

6.16.3.6 vSphere Provision on the Flatcar Linux OS

When provisioning onto the Flatcar Container Linux distribution, you must instruct the bootstrap cluster to make some changes related to the installation paths. To accomplish this, add the `--os-hint flatcar` flag to the `dkp create cluster` command.

Flatcar default network interface name might require specifying. It is most likely to be `ens192` requiring passing the parameter `--virtual-ip-interface ens192` to the `dkp create cluster vsphere` command. Otherwise, the cluster creation might fail because `kube-vip` can not configure the first control-plane virtual IP.

6.16.3.6.1 Flatcar Linux example

These flags are also shown in context on the Create Cluster page for either air-gapped (see page 1410) or non-air-gapped (see page 1386) environments:

```
dkp create cluster vsphere \
    --cluster-name ${CLUSTER_NAME} \
    --os-hint flatcar
```

For provisioning DKP on Flatcar, DKP configures cluster nodes to use Control Groups (cgroups) version 1[988]. In versions prior to Flatcar 3033.3.x, a restart is required in order to apply the changes to the kernel. For more information, refer to the Flatcar documentation[989].

Also note that once Ignition[990] runs, it is not available on reboot.

6.16.3.6.2 Next Topic

Configure MetalLB for a vSphere infrastructure (see page 1376)

988 https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v1/cgroups.html#what-are-cgroups
989 https://www.flatcar.org/docs/latest/container-runtimes/switching-to-unified-cgroups/#starting-new-nodes-with-legacy-cgroups
990 https://www.flatcar.org/docs/latest/provisioning/ignition/#ignition-only-runs-once

6.16.3.7 Configure MetalLB for a vSphere infrastructure

6.16.3.7.1 Create a MetalLB configmap for your vSphere infrastructure.

It is recommended that an external load balancer(LB) be the control plane endpoint. To distribute request load among the control plane machines, configure the load balancer to send requests to all the control plane machines. Configure the load balancer to send requests only to control plane machines that are responding to API requests. If you do not have one, you can use Metal LB to create a MetalLB configmap for your vSphere infrastructure.

Choose one of the following two protocols you want to use to announce service IPs. If your environment is not currently equipped with a load balancer, you can use MetalLB. Otherwise, your own load balancer will work and you can continue the installation process.

To use MetalLB, create a MetalLB configMap for your Pre-provisioned infrastructure. MetalLB uses one of two protocols for exposing Kubernetes services:

- Layer 2, with Address Resolution Protocol (ARP)
- Border Gateway Protocol (BGP

Select one of the following procedures to create your MetalLB manifest for further editing.

6.16.3.7.2 Layer 2 Configuration

Layer 2 mode is the simplest to configure: in many cases, you don't need any protocol-specific configuration, only IP addresses.

Layer 2 mode does not require the IPs to be bound to the network interfaces of your worker nodes. It works by responding to ARP requests on your local network directly, to give the machine's MAC address to clients.

- MetalLB IP address ranges/CIDRs should be within the node's primary network subnet (see page 1065).
- MetalLB IP address ranges/CIDRs and node subnet should not conflict with the Kubernetes cluster pod and service subnets.

For example, the following configuration gives MetalLB control over IPs from 192.168.1.240 to 192.168.1.250, and configures Layer 2 mode:

The following values are generic, enter your specific values into the fields where applicable.

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
EOF
```

When complete, run the following `kubectl` command.

```
kubectl apply -f metallb-conf.yaml
```

6.16.3.7.3 BGP configuration

BGP, like any routing protocol, requires a mesh, so you configure BGP between the nodes. If you are in a Data Center, peer with the BGP routers. The instructions for use with Cloud providers is the same. For a standard configuration featuring one BGP router and one IP address range, you need four pieces of information:

- The router IP address that MetalLB should connect to,
- The router's AS number,
- The AS number MetalLB should use,
- An IP address range expressed as a CIDR prefix.

As an example, if you want to give MetalLB the range 192.168.10.0/24 and AS number 64500, and connect it to a router at 10.0.0.1 with AS number 64501, your configuration will look like:

The following values are generic, enter your specific values into the fields where applicable.

Extract the kubeconfig and deploy a config map for MetalLB using the following command:

```
dkp get kubeconfig -c ${DKP_CLUSTER_NAME} > ${DKP_CLUSTER_NAME}.conf
```

Deploy MetalLB Configuration with the command below:

```
cat << EOF > metallb-conf.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
```

```
  name: config
data:
  config: |
    peers:
    - peer-address: 10.0.0.1
      peer-asn: 64501
      my-asn: 64500
    address-pools:
    - name: default
      protocol: bgp
      addresses:
      - 192.168.10.0/24
EOF
```

When complete, run the following `kubectl` command:

```
kubectl apply -f metallb-conf.yaml
```

6.16.3.7.4 Next Topic

vSphere Output Directory YAML (see page 1378)

6.16.3.8 vSphere Output Directory YAML

You can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag used with `--output=json|yaml`. You create the directory of where to output resources to files.

Using this flag will create multiple files in the specified directory which must already exist:

```
dkp create cluster vsphere
    --cluster-name=${CLUSTER_NAME} \
    --dry-run \
    --output=yaml \
    --output-directory=<existing-directory>
```

For more information regarding this flag or others, please refer to the CLI section of the documentation for the dkp create cluster (see page 1853) command and select your provider.

6.16.3.8.1 Next Step

If none of these custom choices apply, proceed to either section:

- Install vSphere in a Non-air-gapped Environment (see page 1379)
- Install vSphere in an Air-Gapped Environment (see page 1404)

6.16.4 Install vSphere in a Non-air-gapped Environment

In an environment with access to the Internet, you retrieve artifacts from specialized repositories dedicated to them, such as Docker® images contained in DockerHub, and Helm™ Charts that come from a dedicated Helm Chart repository. However, in an air-gapped environment, you need local repositories to store Helm charts, Docker images, and other artifacts. Tools such as JFrog™, Harbor™, and Nexus™ handle multiple types of artifacts in one local repository.

If desired, a local registry can also be used in a non-air-gapped environment for purposes of speed and security. To do so, add the following steps to your non-air-gapped installation process:

- vSphere Registry Mirrors (see page 1370)
- vSphere Load the Registry (see page 1371)

This section provides the instructions to create a Kubernetes cluster in a private network with no access to the Internet. Complete the list of Prerequisites (see page 1356) before you begin the procedures in this section.

- vSphere Create a VM Template (see page 1379)
- vSphere Bootstrap (see page 1384)
- vSphere Create a New Cluster (see page 1386)
- vSphere Make the Cluster Self-Managed (see page 1394)
- vSphere Explore a Cluster (see page 1397)
- vSphere Install Kommander in a Non-air-gapped Environment (see page 1401)

6.16.4.1 vSphere Create a VM Template

You must have at least one image before creating a new cluster. As long as you have an image, this step in your configuration is not required each time since that image can be used to spin up a new cluster. However, if you need different images for different environments or providers, you will need to create a new custom image.

The Konvoy Image Builder (see page 1652) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. Using KIB, you can build an image without requiring access to the internet by providing an additional offline (see page 1674) `--override` flag. You can use these overrides files to customize some of the components installed on your machine image. For example, you could tell KIB to install the FIPS versions of the Kubernetes components.

6.16.4.1.1 Create a vSphere Template for Your Cluster from a Base OS Image

Using the base OS image created in a previous procedure, DKP creates the new vSphere template directly on the vCenter server.

1. Set the following vSphere environment variables on the bastion VM host:

```
export VSPHERE_SERVER=your_vCenter_APIserver_URL
export VSPHERE_USERNAME=your_vCenter_user_name
export VSPHERE_PASSWORD=your_vCenter_password
```

2. Copy the base OS image file created in the vSphere Client to your desired location on the bastion VM host and make a note of the path and file name.
3. Create an `image.yaml` file and add the following variables for vSphere. DKP uses this file and these variables as inputs in the next step. To customize your `image.yaml` file, refer to this section: Customize your Image (see page 1661).
   ⚠ **NOTE**: This example is Ubuntu 20.04. You will need to replace OS name below based on your OS. See other default YAML examples[991] for copy and paste below last step.

```
---
download_images: true
build_name: "ubuntu-2004"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-
init-vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: "<VSPHERE_CLUSTER_NAME>"
  datacenter: "<VSPHERE_DATACENTER_NAME>"
  datastore: "<VSPHERE_DATASTORE_NAME>"
  folder: "<VSPHERE_FOLDER>"
  insecure_connection: "false"
  network: "<VSPHERE_NETWORK>"
  resource_pool: "<VSPHERE_RESOURCE_POOL>"
  template: "os-qualification-templates/d2iq-base-Ubuntu-20.04" # change
default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "ubuntu2004-64"
  # goss params
  distribution: "ubuntu"
  distribution_version: "20.04"
```

991 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

```
# Use following overrides to select the authentication method that can be used
with base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key
will be ignored
```

4. Create a vSphere VM template with your variation of the following command:

   ```
   konvoy-image build images/ova/<image.yaml>
   ```

   Any additional configurations can be added to this command using `--overrides` flags as shown below:

   a. Any credential overrides: `--overrides overrides.yaml`
   b. for FIPS, add this flag: `--overrides overrides/fips.yaml`
   c. for air-gapped, add this flag: `--overrides overrides/offline-fips.yaml`

5. The Konvoy Image Builder (see page 1652) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. This template contains the required artifacts needed to create a Kubernetes cluster.
   When KIB provisions the OS image[992] successfully, it creates a manifest file. The `artifact_id` field of this file contains the name of the AMI ID (AWS), template name (vSphere), or image name (GCP/Azure), for example:

   ```
   {
         "name": "vsphere-clone",
         "builder_type": "vsphere-clone",
         "build_time": 1644985039,
         "files": null,
         "artifact_id": "konvoy-ova-vsphere-rhel-84-1.21.6-1644983717",
         "packer_run_uuid": "260e8110-77f8-ca94-e29e-ac7a2ae779c8",
         "custom_data": {
           "build_date": "2022-02-16T03:55:17Z",
           "build_name": "vsphere-rhel-84",
           "build_timestamp": "1644983717",
           [...]
         }
       }
   ```

992 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

Recommendation: Now we can now see the template created in our vCenter, it is best to rename it to `dkp-<DKP_VERSION>-k8s-<K8S_VERSION>-<DISTRO>`, like `dkp-2.4.0-k8s-1.24.6-ubuntu` to keep templates organized.

6. Next steps are to deploy a DKP cluster using your vSphere template.

Create a Kubernetes Bootstrap Cluster to enable creating your vSphere cluster and moving CAPI objects to it.

6.16.4.1.2 Specific to VMware Cloud Director

For storage, the VCD **Provider** must follow Create a Base OS image in vSphere vCenter (see page 1366) when they create the KIB base image. In VCD, the base image determines the minimum storage available for the VM.

6.16.4.1.2.1 Additional OS YAML files for copy/paste:

RHEL 8.6

```
---
download_images: true
build_name: "rhel-86"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false" # can be "true"
  network: "<VSPHERE_NETWORK>"
  resource_pool: "<VSPHERE_RESOURCE_POOL>"
  template: "os-qualification-templates/d2iq-base-RHEL-86" # change default value
with your base template name
  vsphere_guest_os_type: "rhel8_64Guest"
  guest_os_type: "rhel8-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "8.6"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
```

```
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Rocky Linux 9.1

```
---
download_images: true
build_name: "rocky-91"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "os-qualification-templates/d2iq-base-RockyLinux-9.1" # change default
 value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "rocky9-64"
  # goss params
  distribution: "rocky"
  distribution_version: "9.1"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

YAML Files (see page 1658) found in Konvoy Image Builder section.

6.16.4.1.2.2 Next Step

vSphere Bootstrap (see page 1384)

6.16.4.2 vSphere Bootstrap

6.16.4.2.1 Prepare to deploy Kubernetes clusters

To create Kubernetes clusters, Konvoy uses Cluster API[993] (CAPI) controllers, which run on a Kubernetes cluster. To get started creating your vSphere cluster, you need a *bootstrap* cluster. By default, Konvoy creates a bootstrap cluster for you in a Docker container using the Kubernetes-in-Docker (KIND[994]) tool.

6.16.4.2.2 Prerequisites

Before you begin, you must:

- Ensure the `dkp` binary can be found in your $PATH.
- Complete the steps in Create a CAPI VM template (see page 1379)

6.16.4.2.3 Bootstrap Cluster Lifecycle Services

1. Review Universal Configurations for all Infrastructure Providers (see page 1052) regarding settings, flags and other choices and then begin bootstrapping.
2. Create a bootstrap cluster:

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config
   ```

   - Configuring an HTTP/HTTPS Proxy (see page 1053) use `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful.

HTTP Only:

To create a bootstrap cluster in a proxied environment (see page 1054) use this command syntax, in addition to any other flags you may need:

```
--http-proxy <string> \
--https-proxy <string> \
--no-proxy <string>
```

3. The output resembles this example:

   ```
   ✓ Creating a bootstrap cluster
   ✓ Initializing new CAPI components
   ```

993 https://cluster-api.sigs.k8s.io/
994 https://github.com/kubernetes-sigs/kind

Konvoy creates a bootstrap cluster using KIND[995] as a library. Konvoy then deploys the following Cluster API[996] providers on the cluster:

- Core Provider[997]
- vSphere Infrastructure Provider[998]
- Kubeadm Bootstrap Provider[999]
- Kubeadm ControlPlane Provider[1000]

4. Ensure that the CAPV controllers are present with the command:

```
kubectl get pods -n capv-system
```

The output resembles the following:

```
NAME                                      READY   STATUS    RESTARTS   AGE
capv-controller-manager-785c5978f-nnfns   1/1     Running   0          13h
```

5. Konvoy waits until the controller-manager and webhook deployments of these providers are ready. List these deployments using this command:

```
kubectl get --all-namespaces deployments -l=clusterctl.cluster.x-k8s.i
```

The output resembles the following:

```
NAMESPACE                           NAME
READY   UP-TO-DATE   AVAILABLE   AGE
capa-system                         capa-controller-manager
1/1     1            1           22m
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-manager
1/1     1            1           22m
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-
manager   1/1     1            1           22m
capi-system                         capi-controller-manager
1/1     1            1           22m
cappp-system                        cappp-controller-manager
1/1     1            1           22m
capv-system                         capv-controller-manager
1/1     1            1           22m
```

995 https://github.com/kubernetes-sigs/kind
996 https://cluster-api.sigs.k8s.io/
997 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/
998 https://github.com/kubernetes-sigs/cluster-api-provider-vsphere
999 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/bootstrap/kubeadm
1000 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/controlplane/kubeadm

```
capz-system                        capz-controller-manager
1/1     1            1           22m
cert-manager                       cert-manager
1/1     1            1           22m
cert-manager                       cert-manager-cainjector
1/1     1            1           22m
cert-manager                       cert-manager-webhook
1/1     1            1           22m
```

6.16.4.2.4 Next Step

vSphere Create a New Cluster (see page 1386)

6.16.4.3 vSphere Create a New Cluster

6.16.4.3.1 Prerequisites

Before you begin, make sure you have created a vSphere Bootstrap (see page 1384) cluster.

6.16.4.3.1.1 Name your Cluster

1. Give your cluster a unique name suitable for your environment. The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[1001] for more naming information.
2. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=<my-vsphere-cluster>
```

6.16.4.3.1.2 Create a New vSphere Kubernetes Cluster

Follow these steps:

1. Use the following command to set the environment variables for vSphere:

```
export VSPHERE_SERVER=<example.vsphere.url>
export VSPHERE_USERNAME=<user@example.vsphere.url>
export VSPHERE_PASSWORD=<example_password>
```

1001 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

2. Ensure your vSphere credentials are up-to-date by refreshing the credentials with the command:

```
dkp update bootstrap credentials vsphere
```

3. Generate the Kubernetes cluster objects by copying and editing this command to include the correct values, including the VM template name you assigned in the previous procedure:

- To increase Dockerhub's rate limit[1002] use your Dockerhub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username= --registry-mirror-password=` on the `dkp create cluster command`.
- ⚠ **IMPORTANT:** Ensure your subnets (see page 1065) do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

- The following example shows a common configuration. Other options and their corresponding flags are available in the expands below the code and were also explained in vSphere Cluster Creation Customization Choices (see page 1368).
  - See dkp create cluster vsphere (see page 1861) reference for the full list of cluster creation flags.

```
dkp create cluster vsphere \
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <xxx.yyy.zzz.000> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file <SSH_PUBLIC_KEY_FILE> \
```

1002 https://docs.docker.com/docker-hub/download-rate-limit/

```
  --resource-pool <RESOURE_POOL_NAME> \
  --virtual-ip-interface <ip_interface_name> \
  --vm-template <TEMPLATE_NAME>
  --dry-run \
  --output=yaml \
  > ${CLUSTER_NAME}.yaml
```

Expand the drop-downs for **registry, HTTP, FIPS and other flags** to use during the cluster creation step above. For more information regarding this flag or others, please refer to the CLI section of the documentation for dkp create cluster (see page 1853) and select your provider.

Flatcar

Flatcar OS (see page 1375) use this flag used to instruct the bootstrap cluster to make some changes related to the installation paths:

```
--os-hint flatcar
```

If using a REGISTRY MIRROR, use these FLAGS in your create cluster command:

(Optional) Use a registry mirror (see page 1609). Configure your cluster to use an existing local registry (see page 1606) as a mirror when attempting to pull images previously pushed to your registry (see page 317).

Export the environment variable with your registry information. These tell the cluster where to locate the local registry to use by defining the URL or other needed information:

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_CA=<path to the CA on the bastion>
export REGISTRY_USERNAME=<username>
export REGISTRY_USERNAME=<password>
```

- `REGISTRY_URL` : the address of an existing container registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- `REGISTRY_CA` : (optional) the path on the bastion machine to the container registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.

When **creating the cluster**, apply the variables you defined above during the `dkp create cluster` command with the flags needed for your environment:

```
--registry-mirror-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
```

```
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD}
```

See also: vSphere Registry Mirrors (see page 1370) Registry Mirror Tools (see page 1606)

FIPS Requirements

To create a cluster in FIPS mode (see page 1598), inform the controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes by adding those flags to `dkp create cluster` command:

```
--kubernetes-version=v1.28.7+fips.0 \
--etcd-version=3.5.10+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
```

HTTP ONLY

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

```
--http-proxy <<http proxy list>>
--https-proxy <<https proxy list>>
--no-proxy <<no proxy list>>
```

- To configure the Control Plane and Worker nodes to use an HTTP proxy:

  ```
  export CONTROL_PLANE_HTTP_PROXY=http://example.org:8080
  export CONTROL_PLANE_HTTPS_PROXY=http://example.org:8080
  export
  CONTROL_PLANE_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1
  ,10.96.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.s
  vc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.sv
  c.cluster,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"

  export WORKER_HTTP_PROXY=http://example.org:8080
  export WORKER_HTTPS_PROXY=http://example.org:8080
  export
  WORKER_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96.
  0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kube
  rnetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.clust
  er,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"
  ```

- Replace:
  - `example.org,example.com,example.net` with you internal addresses
  - `localhost` and `127.0.0.1` addresses should not use the proxy
  - `10.96.0.0/12` is the default Kubernetes service subnet

  - `192.168.0.0/16` is the default Kubernetes pod subnet
  - `kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local` is the internal Kubernetes kube-apiserver service
  - `.svc,.svc.cluster,.svc.cluster.local` is the internal Kubernetes services
- Create a Kubernetes cluster with HTTP proxy configured using the flags and exported variables during the `dkp create cluster` command:

```
--control-plane-http-proxy=${CONTROL_PLANE_HTTP_PROXY} \
--control-plane-https-proxy=${CONTROL_PLANE_HTTPS_PROXY} \
--control-plane-no-proxy=${CONTROL_PLANE_NO_PROXY} \
--worker-http-proxy=${WORKER_HTTP_PROXY} \
--worker-https-proxy=${WORKER_HTTPS_PROXY} \
--worker-no-proxy=${WORKER_NO_PROXY} \
```

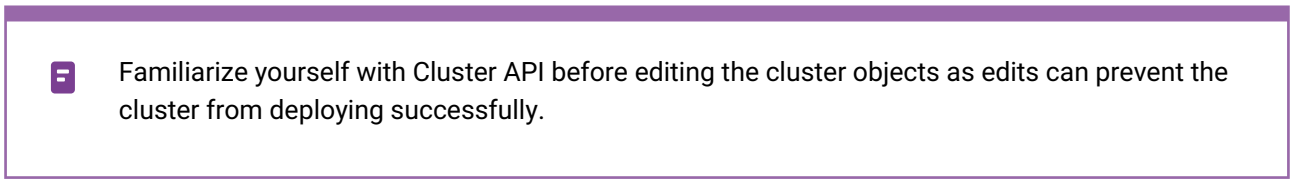
Individual manifests using the Output Directory flag:

You can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag. This will create multiple files in the specified directory which must already exist:

```
--output-directory=<existing-directory>
```

Refer to the Cluster Creation Customization Choices (see page 1170) section for more information on how to use optional flags such as the `--output-directory` flag.

4. Inspect or edit the cluster objects and familiarize yourself with Cluster API before editing the cluster objects as edits can prevent the cluster from deploying successfully. See Customizing CAPI Components for a Cluster (see page 1061).



5. **(Optional)** Modify Control Plane Audit logs - Users can make modifications to the `KubeadmControlplane` cluster-api object to configure different `kubelet` options. See Configuring the Control Plane (see page 1613) if you wish to configure your control plane beyond the existing options that are available from flags.
6. Create the cluster from the objects generated in the `dry run`. A warning will appear in the console if the resource already exists and will require you to remove the resource or update your YAML.

```
kubectl create -f ${CLUSTER_NAME}.yaml
```

NOTE: If you used the `--output-directory` flag in your `dkp create .. --dry-run` step above, create the cluster from the objects you created by specifying the directory:

```
kubectl create -f <existing-directory>/
```

OUTPUT:

```
cluster.cluster.x-k8s.io/vsphere-example created
cluster.infrastructure.cluster.x-k8s.io/vsphere-example created
kubeadmcontrolplane.controlplane.cluster.x-k8s.io/vsphere-example-control-plane
created
machinedeployment.cluster.x-k8s.io/vsphere-example-mp-0 created
kubeadmconfigtemplate.bootstrap.cluster.x-k8s.io/vsphere-example-mp-0 created
```

7. Use the `wait` command to monitor the cluster control-plane readiness:

   ```
   kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
   timeout=20m
   ```

   Output:

   ```
   cluster.cluster.x-k8s.io/${CLUSTER_NAME} condition met
   ```

   The `READY` status becomes `True` after the cluster control-plane becomes Ready in one of the following steps.

8. After DKP creates the objects on the API server, the Cluster API controllers reconcile them, creating infrastructure and machines. As the controllers progress, they update the Status of each object. Run the DKP describe command to monitor the current status of the cluster:

   ```
   dkp describe cluster -c ${CLUSTER_NAME}
   ```

OUTPUT:

```
NAME                                                                READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/d2iq-e2e-cluster_name-1                                     True
13h
├─ClusterInfrastructure - VSphereCluster/d2iq-e2e-cluster_name-1    True
13h
```

```
├─ControlPlane - KubeadmControlPlane/d2iq-control-plane            True
13h
│ ├─Machine/d2iq--control-plane-7llgd                              True
13h
│ ├─Machine/d2iq--control-plane-vncbl                              True
13h
│ └─Machine/d2iq--control-plane-wbgrm                              True
13h
└─Workers
    └─MachineDeployment/d2iq--md-0                                 True
13h
    ├─Machine/d2iq--md-0-74c849dc8c-67rv4                          True
13h
    ├─Machine/d2iq--md-0-74c849dc8c-n2skc                          True
13h
    ├─Machine/d2iq--md-0-74c849dc8c-nkftv                          True
13h
    └─Machine/d2iq--md-0-74c849dc8c-sqklv                          True
13h
```

9. Check all machines has `NODE_NAME` assigned:

```
kubectl get machines
```

OUTPUT:

```
NAME                                      CLUSTER              NODENAME
PROVIDERID                                      PHASE     AGE   VERSION
d2iq-e2e-cluster-1-control-plane-7llgd    d2iq-e2e-cluster-1   d2iq-e2e-cluster-1-
control-plane-7llgd    vsphere://421638e2-e776-9af6-f683-5e105de5da5a   Running
13h   v1.28.7
d2iq-e2e-cluster-1-control-plane-vncbl    d2iq-e2e-cluster-1   d2iq-e2e-cluster-1-
control-plane-vncbl    vsphere://42168835-7fef-95c4-3652-ebcad3e10d36   Running
13h   v1.28.7
d2iq-e2e-cluster-1-control-plane-wbgrm    d2iq-e2e-cluster-1   d2iq-e2e-cluster-1-
control-plane-wbgrm    vsphere://421642df-afc4-b6c2-9e61-5b86e7c37eac   Running
13h   v1.28.7
d2iq-e2e-cluster-1-md-0-74c849dc8c-67rv4   d2iq-e2e-cluster-1   d2iq-e2e-cluster-1-
md-0-74c849dc8c-67rv4   vsphere://4216f467-8483-73cb-a8b6-8d6a4a71e4b4   Running
14h   v1.28.7
d2iq-e2e-cluster-1-md-0-74c849dc8c-n2skc   d2iq-e2e-cluster-1   d2iq-e2e-cluster-1-
md-0-74c849dc8c-n2skc   vsphere://42161cde-9904-4dd2-7a3e-cdfc7655f090   Running
14h   v1.28.7
d2iq-e2e-cluster-1-md-0-74c849dc8c-nkftv   d2iq-e2e-cluster-1   d2iq-e2e-cluster-1-
md-0-74c849dc8c-nkftv   vsphere://42163a0d-eb8d-b5a6-82d5-188e24817c00   Running
14h   v1.28.7
d2iq-e2e-cluster-1-md-0-74c849dc8c-sqklv   d2iq-e2e-cluster-1   d2iq-e2e-cluster-1-
md-0-74c849dc8c-sqklv   vsphere://42161dff-92a5-6da9-7ac1-e987e2c8fed2   Running
14h   v1.28.7
```

10. Verify that the kubeadm control plane is ready with the command:

```
kubectl get kubeadmcontrolplane
```

The output appears similar to the following:

```
NAME                                CLUSTER             INITIALIZED   API
SERVER AVAILABLE   REPLICAS   READY   UPDATED   UNAVAILABLE   AGE   VERSION
d2iq-e2e-cluster-1-control-plane   d2iq-e2e-cluster-1   true          true
3          3       3         0             14h   v1.28.7
```

11. Describe the kubeadm control plane and check its status and events with the command:

```
kubectl describe kubeadmcontrolplane
```

12. As they progress, the controllers also create Events, which you can list using the command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, this example uses `grep`. You can also use separate commands to get Events for specific objects, such as `kubectl get events --field-selector involvedObject.kind="VSphereCluster"` and `kubectl get events --field-selector involvedObject.kind="VSphereMachine"`.

DKP uses the vsphere CSI driver as the default storage provider (see page 110). Use a Kubernetes CSI[1003] compatible storage that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[1004] for more information.
If you're not using the default, you cannot deploy an alternate provider until ***after*** the `dkp create cluster` is finished. However, it must be determined before Kommander installation.

6.16.4.3.1.3 Known Limitations

Be aware of these limitations in the current release of DKP Konvoy.

1003 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
1004 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

- The DKP Konvoy version used to create a bootstrap cluster must match the DKP Konvoy version used to create a workload cluster.
- DKP Konvoy supports deploying one workload cluster.
- DKP Konvoy generates a set of objects for one Node Pool.
- DKP Konvoy does not validate edits to cluster objects.

The optional next step is to Make the vSphere Cluster Self-managed (see page 1394). The step is optional because, as an example, if you are using an existing, self-managed cluster to create a managed cluster, you would not want the managed cluster to be self-managed.

6.16.4.3.1.4 Next Steps

vSphere Make the Cluster Self-Managed (see page 1394)

6.16.4.4 vSphere Make the Cluster Self-Managed

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which then deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, which makes the workload cluster self-managed. This section describes how to make a workload cluster self-managed.

> Before starting, ensure you create a workload cluster as described in Create a New vSphere Cluster or Create a New Air-gapped vSphere Cluster.

This page contains instructions on how to make your cluster self-managed. This is necessary if there is only one cluster in your environment, or if this cluster should become the Management cluster in a multi-cluster environment.

> If you already have a self-managed or Management cluster in your environment, skip this page.

6.16.4.4.1 Make the New Kubernetes Cluster Manage Itself

To create a Management Cluster or stand alone Essential Cluster, the cluster must manage itself. Follow these steps to achieve:

1. Deploy cluster lifecycle services on the workload cluster:

   ```
   dkp create capi-components --kubeconfig ${CLUSTER_NAME}.conf
   ```

   Output:

```
✓ Initializing new CAPI components
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

2. Move the Cluster API objects from the bootstrap to the workload cluster:
   The cluster lifecycle services on the workload cluster are ready, but the workload cluster configuration is on the bootstrap cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the bootstrap to the workload cluster. This process is also called a Pivot[1005].

   ```
   dkp move capi-resources --to-kubeconfig ${CLUSTER_NAME}.conf
   ```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

Output:

```
 ✓ Moving cluster resources
You can now view resources in the moved cluster by using the --kubeconfig flag with
kubectl. For example: kubectl --kubeconfig=vsphere-example.conf get nodes
```

To ensure only one set of cluster lifecycle services manages the workload cluster, Konvoy first pauses reconciliation of the objects on the bootstrap cluster, then creates the objects on the workload cluster. As Konvoy copies the objects, the cluster lifecycle services on the workload cluster reconcile the objects. The workload cluster becomes self-managed after Konvoy creates all the objects. If it fails, the move command can be safely retried.

1005 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

3. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf wait --for=condition=ControlPlaneRead
y "clusters/${CLUSTER_NAME}" --timeout=20m
```

Output:

```
cluster.cluster.x-k8s.io/vsphere-example condition met
```

After moving the cluster lifecycle services to the workload cluster, remember to use Konvoy with the workload cluster kubeconfig.

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

```
dkp describe cluster --kubeconfig ${CLUSTER_NAME}.conf -c ${CLUSTER_NAME}
```

Output:

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/vsphere-example-1                                           True
13h
├─ClusterInfrastructure - VSphereCluster/vsphere-example-1          True
13h
├─ControlPlane - KubeadmControlPlane/vsphere-example-control-plane  True
13h
│ ├─Machine/vsphere-example-control-plane-7llgd                     True
13h
│ ├─Machine/vsphere-example-control-plane-vncbl                     True
13h
│ └─Machine/vsphere-example-control-plane-wbgrm                     True
13h
└─Workers
    └─MachineDeployment/vsphere-example-md-0                        True
13h
    ├─Machine/vsphere-example-md-0-74c849dc8c-67rv4                 True
13h
    ├─Machine/vsphere-example-md-0-74c849dc8c-n2skc                 True
13h
    ├─Machine/vsphere-example-md-0-74c849dc8c-nkftv                 True
13h
```

```
    └─Machine/vsphere-example-md-0-74c849dc8c-sqklv                 True
13h
```

5. Remove the bootstrap cluster, if desired, as the workload cluster is now self-managed:

```
dkp delete bootstrap
```

Output:

```
✓ Deleting bootstrap cluster
```

6.16.4.4.1.1 Known Limitations

Be aware of these limitations in the current release of DKP Konvoy.

- Before making a workload cluster self-managed, be sure that its control plane nodes have sufficient permissions for running Cluster API controllers.
- DKP Konvoy supports moving only one set of cluster objects from the bootstrap cluster to the workload cluster, or vice-versa.
- DKP Konvoy only supports moving all namespaces in the cluster; DKP does not support migration of individual namespaces.

6.16.4.4.1.2 Next Step

vSphere Explore a Cluster (see page 1397)

6.16.4.5 vSphere Explore a Cluster

This guide explains how to use the command line interface to interact with your newly-deployed Kubernetes cluster.

Before you start, make sure you have created a workload cluster (see page 1386) and, if needed, that you have made the cluster self-managing (see page 1394).

6.16.4.5.1 Get the `kubeconfig` File for the New Kubernetes Cluster

When the workload cluster is created, the cluster lifecycle services generate a kubeconfig file for the workload cluster, and write it to a *Secret*. The kubeconfig file is scoped to the cluster Administrator.

Get the kubeconfig from the *Secret*, and write it to a file using this command:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

6.16.4.5.1.1 Create a StorageClass with a vSphere datastore

Follow these steps:

1. Access the Datastore tab in the vSphere client and select a datastore by name.
2. Copy the URL of that datastore from the information dialog that displays.
3. Return to the DKP CLI, and delete the existing `StorageClass` with the command:

   ```
   kubectl delete storageclass vsphere-raw-block-sc
   ```

4. Run the following command to create a new StorageClass, supplying the correct values for your environment:

   ```
   cat <<EOF > vsphere-raw-block-sc.yaml
   kind: StorageClass
   apiVersion: storage.k8s.io/v1
   metadata:
     annotations:
       storageclass.kubernetes.io/is-default-class: "true"
     name: vsphere-raw-block-sc
   provisioner: csi.vsphere.vmware.com
   parameters:
     datastoreurl: "<url>"
   volumeBindingMode: WaitForFirstConsumer
   EOF
   ```

6.16.4.5.1.2 Explore nodes and pods in the new cluster

1. List the nodes using this command:

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
   ```

   NOTE: It may take a few minutes for the Status to move to `Ready` while the Pod network is deployed. The Node's Status should change to Ready soon after the `calico-node` DaemonSet Pods are Ready.

   The output resembles the following example:

```
NAME                                        STATUS   ROLES                  AGE
VERSION
d2iq-e2e-cluster-1-control-plane-7llgd      Ready    control-plane,master   20h
v1.27.6
d2iq-e2e-cluster-1-control-plane-vncbl      Ready    control-plane,master   19h
v1.27.6
d2iq-e2e-cluster-1-control-plane-wbgrm      Ready    control-plane,master   19h
v1.27.6
d2iq-e2e-cluster-1-md-0-74c849dc8c-67rv4    Ready    <none>                 19h
v1.27.6
d2iq-e2e-cluster-1-md-0-74c849dc8c-n2skc    Ready    <none>                 19h
v1.27.6
d2iq-e2e-cluster-1-md-0-74c849dc8c-nkftv    Ready    <none>                 19h
v1.27.6
d2iq-e2e-cluster-1-md-0-74c849dc8c-sqklv    Ready    <none>                 19h
v1.27.6
```

2. List the pods using this command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get --all-namespaces pods
```

OUTPUT:

The output resembles the following example:

```
NAMESPACE               NAME
READY   STATUS     RESTARTS      AGE
calico-system           calico-kube-controllers-57fbd7bd59-qqd96
1/1     Running    0             20h
calico-system           calico-node-2m524
1/1     Running    3 (19h ago)   19h
calico-system           calico-node-bbhg5
1/1     Running    0             20h
calico-system           calico-node-cc5lf
1/1     Running    2 (19h ago)   19h
calico-system           calico-node-cwg7x
1/1     Running    1 (19h ago)   19h
calico-system           calico-node-d59hn
1/1     Running    1 (19h ago)   19h
calico-system           calico-node-qmmcz
1/1     Running    0             19h
calico-system           calico-node-wdqhx
1/1     Running    0             19h
calico-system           calico-typha-655489d8cc-b5jnt
1/1     Running    0             20h
calico-system           calico-typha-655489d8cc-q92x9
1/1     Running    0             19h
```

```
calico-system            calico-typha-655489d8cc-vjlkx
1/1     Running    0             19h
kube-system              cluster-autoscaler-68c759fbf6-7d2ck
0/1     Init:0/1   0             20h
kube-system              coredns-78fcd69978-qn4qt
1/1     Running    0             20h
kube-system              coredns-78fcd69978-wqpmg
1/1     Running    0             20h
kube-system              etcd-d2iq-e2e-cluster-1-control-plane-7llgd
1/1     Running    0             20h
kube-system              etcd-d2iq-e2e-cluster-1-control-plane-vncbl
1/1     Running    0             19h
kube-system              etcd-d2iq-e2e-cluster-1-control-plane-wbgrm
1/1     Running    0             19h
kube-system              kube-apiserver-d2iq-e2e-cluster-1-control-plane-7llgd
1/1     Running    0             20h
kube-system              kube-apiserver-d2iq-e2e-cluster-1-control-plane-vncbl
1/1     Running    0             19h
kube-system              kube-apiserver-d2iq-e2e-cluster-1-control-plane-wbgrm
1/1     Running    0             19h
kube-system              kube-controller-manager-d2iq-e2e-cluster-1-control-
plane-7llgd   1/1     Running    1 (19h ago)   20h
kube-system              kube-controller-manager-d2iq-e2e-cluster-1-control-plane-
vncbl   1/1     Running    0             19h
kube-system              kube-controller-manager-d2iq-e2e-cluster-1-control-plane-
wbgrm   1/1     Running    0             19h
kube-system              kube-proxy-cpscs
1/1     Running    0             19h
kube-system              kube-proxy-hhmxq
1/1     Running    0             19h
kube-system              kube-proxy-hxhnk
1/1     Running    0             19h
kube-system              kube-proxy-nsrbp
1/1     Running    0             19h
kube-system              kube-proxy-scxfg
1/1     Running    0             20h
kube-system              kube-proxy-tth4k
1/1     Running    0             19h
kube-system              kube-proxy-x2xfx
1/1     Running    0             19h
kube-system              kube-scheduler-d2iq-e2e-cluster-1-control-plane-7llgd
1/1     Running    1 (19h ago)   20h
kube-system              kube-scheduler-d2iq-e2e-cluster-1-control-plane-vncbl
1/1     Running    0             19h
kube-system              kube-scheduler-d2iq-e2e-cluster-1-control-plane-wbgrm
1/1     Running    0             19h
kube-system              kube-vip-d2iq-e2e-cluster-1-control-plane-7llgd
1/1     Running    1 (19h ago)   20h
kube-system              kube-vip-d2iq-e2e-cluster-1-control-plane-vncbl
1/1     Running    0             19h
kube-system              kube-vip-d2iq-e2e-cluster-1-control-plane-wbgrm
1/1     Running    0             19h
```

```
kube-system              vsphere-cloud-controller-manager-4zj7q
1/1     Running    0             19h
kube-system              vsphere-cloud-controller-manager-87tgm
1/1     Running    0             19h
kube-system              vsphere-cloud-controller-manager-xqmn4
1/1     Running    1 (19h ago)   20h
node-feature-discovery   node-feature-discovery-master-84c67dcbb6-txfw9
1/1     Running    0             20h
node-feature-discovery   node-feature-discovery-worker-8tg2l
1/1     Running    3 (19h ago)   19h
node-feature-discovery   node-feature-discovery-worker-c5f6q
1/1     Running    0             19h
node-feature-discovery   node-feature-discovery-worker-fjfkm
1/1     Running    0             19h
node-feature-discovery   node-feature-discovery-worker-x6tz8
1/1     Running    0             19h
tigera-operator          tigera-operator-d499f5c8f-r2srj
1/1     Running    1 (19h ago)   20h
vmware-system-csi        vsphere-csi-controller-7ffd6884cc-d7rql
7/7     Running    5 (19h ago)   20h
vmware-system-csi        vsphere-csi-controller-7ffd6884cc-k82cm
7/7     Running    2 (19h ago)   20h
vmware-system-csi        vsphere-csi-controller-7ffd6884cc-qttkp
7/7     Running    1 (19h ago)   20h
vmware-system-csi        vsphere-csi-node-678hw
3/3     Running    0             19h
vmware-system-csi        vsphere-csi-node-6tbsh
3/3     Running    0             19h
vmware-system-csi        vsphere-csi-node-9htwr
3/3     Running    5 (20h ago)   20h
vmware-system-csi        vsphere-csi-node-g8r6l
3/3     Running    0             19h
vmware-system-csi        vsphere-csi-node-ghmr6
3/3     Running    0             19h
vmware-system-csi        vsphere-csi-node-jhvgm
3/3     Running    0             19h
vmware-system-csi        vsphere-csi-node-rp77r
3/
```

6.16.4.5.1.3 Next Step

vSphere Install Kommander in a Non-air-gapped Environment (see page 1401)

6.16.4.6 vSphere Install Kommander in a Non-air-gapped Environment

This section provides installation instructions for the Kommander component of DKP in a non-air-gapped vSphere environment.

6.16.4.6.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

6.16.4.6.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init > kommander.yaml
   ```

4. If required, customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include:
   - Custom Domains and Certificates
   - HTTP proxy
   - Disabling the AI Navigator application
   - External Load Balancer
   - GPU utilization, etc.
   - Rook Ceph customization for Pre-provisioned environments (see page 1541)

5. *If required:* If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

   ```
   ...
   apps:
     traefik:
       enabled: true
       values: |
         service:
           annotations:
   ```

```
          service.beta.kubernetes.io/aws-load-balancer-internal: "true
...
```

6. Expand **one** of the following sets of instructions, depending on your license and application environments:

Essential License: Install Kommander in a Non-Air-Gapped Environment

6.16.4.6.1.2 Essential License: Install Kommander

Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Enterprise License: Install Kommander in a Non-air-gapped Environment with DKP Catalog Applications

6.16.4.6.1.3 Enterprise License: Install Kommander with DKP Catalog Applications

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.7.0
...
```

If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

6.16.4.6.2 Kommander Customizations

You can configure the Kommander component of DKP during the initial installation, and also post-installation using the DKP CLI. If you are not sure of what you want to customize during install, then proceed to the next step. To read about Kommander component customization options, refer to this section of the documetation: Kommander Customizations (see page 1558)

6.16.4.6.3 Next Step

Day 2 - Cluster Operations Management (see page 590)

6.16.5 Install vSphere in an Air-Gapped Environment

6.16.5.1 Create a Kubernetes vSphere cluster in a private network with no access to the Internet (air-gapped)

In an environment with access to the Internet, you retrieve artifacts from specialized repositories dedicated to them, such as Docker® images contained in DockerHub, and Helm™ Charts that come from a dedicated Helm Chart repository. However, in an air-gapped environment, you need local repositories to store Helm charts, Docker images, and other artifacts. Tools such as JFrog™, Harbor™, and Nexus™ handle multiple types of artifacts in one local repository.

This section provides the instructions to create a Kubernetes cluster in a private network with no access to the Internet. Complete the list of Prerequisites (see page 1356) before you begin the procedures in this section.

- vSphere Create an Air-gapped CAPI VM Template (see page 1404)
- vSphere Air-gapped Seed the Registry (see page 1408)
- vSphere Create an Air-gapped Bootstrap Cluster (see page 1409)
- vSphere Create a New Air-gapped Cluster (see page 1410)
- vSphere Make your Air-gapped Cluster Self-Managed (see page 1416)
- vSphere Explore your Air-gapped Cluster (see page 1419)
- vSphere Install Kommander in an Air-gapped Environment (see page 1424)

6.16.5.2 vSphere Create an Air-gapped CAPI VM Template

You must have at least one image before creating a new cluster. As long as you have an image, this step in your configuration is not required each time since that image can be used to spin up a new cluster. However, if you need different images for different environments or providers, you will need to create a new custom image.

The Konvoy Image Builder (see page 1652) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. Using KIB, you can build an image without requiring access to the internet by providing an additional offline (see page 1674) `--override` flag. You can use these overrides files to customize some of the components installed on your machine image. For example, you could tell KIB to install the FIPS versions of the Kubernetes components.

Create a vSphere template for your cluster from a base OS image

Using KIB (see page 1626), you can build an image without requiring access to the internet by providing an additional `--override` flag.

1. Assuming you have downloaded (see page 76) `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz`, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz && cd dkp-v2.8.0/kib
   ```

2. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.
3. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle`. This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml`. Example command:

   ```
   ./konvoy-image create-package-bundle --os redhat-8.4 --output-
   directory=artifacts
   ```

   NOTE: For FIPS, pass the flag: `--fips`

   NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY`. Example command:

   ```
   export RHSM_ACTIVATION_KEY="-ci"
   export RHSM_ORG_ID="1232131"
   ```

4. Follow the instructions to build a vSphere template below and if applicable, set the override `--overrides overrides/offline.yaml` flag described in Step 4 below.

6.16.5.2.1 Create a vSphere Template for Your Cluster from a Base OS Image

Using the base OS image created in a previous procedure, DKP creates the new vSphere template directly on the vCenter server.

1. Set the following vSphere environment variables on the bastion VM host:

```
export VSPHERE_SERVER=your_vCenter_APIserver_URL
export VSPHERE_USERNAME=your_vCenter_user_name
export VSPHERE_PASSWORD=your_vCenter_password
```

2. Copy the base OS image file created in the vSphere Client to your desired location on the bastion VM host and make a note of the path and file name.
3. Create an `image.yaml` file and add the following variables for vSphere. DKP uses this file and these variables as inputs in the next step. To customize your `image.yaml` file, refer to this section: Customize your Image (see page 1661).
   ⚠ **NOTE**: This example is Ubuntu 20.04. You will need to replace OS name below based on your OS. See other default YAML examples[1006] for copy and paste below last step.

```
---
download_images: true
build_name: "ubuntu-2004"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-
init-vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: "<VSPHERE_CLUSTER_NAME>"
  datacenter: "<VSPHERE_DATACENTER_NAME>"
  datastore: "<VSPHERE_DATASTORE_NAME>"
  folder: "<VSPHERE_FOLDER>"
  insecure_connection: "false"
  network: "<VSPHERE_NETWORK>"
  resource_pool: "<VSPHERE_RESOURCE_POOL>"
  template: "os-qualification-templates/d2iq-base-Ubuntu-20.04" # change
default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "ubuntu2004-64"
  # goss params
  distribution: "ubuntu"
  distribution_version: "20.04"
# Use following overrides to select the authentication method that can be used
with base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key
will be ignored
```

1006 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

4. Create a vSphere VM template with your variation of the following command:

```
konvoy-image build images/ova/<image.yaml>
```

Any additional configurations can be added to this command using `--overrides` flags as shown below:

   a. Any credential overrides: `--overrides overrides.yaml`
   b. for FIPS, add this flag: `--overrides overrides/fips.yaml`
   c. for air-gapped, add this flag: `--overrides overrides/offline-fips.yaml`

5. The Konvoy Image Builder (see page 1652) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. This template contains the required artifacts needed to create a Kubernetes cluster.
When KIB provisions the OS image[1007] successfully, it creates a manifest file. The `artifact_id` field of this file contains the name of the AMI ID (AWS), template name (vSphere), or image name (GCP/Azure), for example:

```
{
      "name": "vsphere-clone",
      "builder_type": "vsphere-clone",
      "build_time": 1644985039,
      "files": null,
      "artifact_id": "konvoy-ova-vsphere-rhel-84-1.21.6-1644983717",
      "packer_run_uuid": "260e8110-77f8-ca94-e29e-ac7a2ae779c8",
      "custom_data": {
        "build_date": "2022-02-16T03:55:17Z",
        "build_name": "vsphere-rhel-84",
        "build_timestamp": "1644983717",
        [...]
      }
    }
```

Recommendation: Now we can now see the template created in our vCenter, it is best to rename it to `dkp-<DKP_VERSION>-k8s-<K8S_VERSION>-<DISTRO>`, like `dkp-2.4.0-k8s-1.24.6-ubuntu` to keep templates organized.

6. Next steps are to deploy a DKP cluster using your vSphere template.

1007 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

6.16.5.2.2 Specific to VMware Cloud Director

For storage, the VCD **Provider** must follow Create an Air-gapped Base OS VM Image[1008] when they create the KIB base image. In VCD, the base image determines the minimum storage available for the VM.

6.16.5.2.2.1 Next Step

vSphere Air-gapped Seed the Registry (see page 1408)

6.16.5.3 vSphere Air-gapped Seed the Registry

Before creating an air-gapped Kubernetes cluster, you need to load the required images in a local registry for the Konvoy component. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1606) page for more information.

1. If not already done in prerequisites, download (see page 76) the air-gapped bundle `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap cluster, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

   ```
   cd dkp-v2.8.1
   ```

3. Set an environment variable with your registry address and any other needed variables using this command:

   ```
   export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   export REGISTRY_CA=<path to the cacert file on the bastion>
   ```

1008 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29918450/Create+an+Air-gapped+Base+OS+VM+Image

4. Execute the following command to load the air-gapped image bundle into your private registry using any of the relevant flags to apply variables above:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --
to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-
registry-password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

6.16.5.3.1 Next Step

vSphere Create an Air-gapped Bootstrap Cluster (see page 1409)

6.16.5.4 vSphere Create an Air-gapped Bootstrap Cluster

6.16.5.4.1 Prerequisites

Before you perform this procedure, ensure that you have created a CAPI VM template (see page 1404).

HTTP Only flags:

To create a bootstrap cluster in a proxied environment (see page 1054) use this command syntax, in addition to any other flags you may need:

```
--http-proxy <string> \
--https-proxy <string> \
--no-proxy <string>
```

- Flatcar OS (see page 1375) use `--os-hint` to instruct the bootstrap cluster to make some changes related to the installation paths:

```
--os-hint flatcar
```

The output resembles this example:

```
✓ Creating a bootstrap cluster
✓ Initializing new CAPI components
```

4. Ensure that the CAPV controllers are present with the command:

```
kubectl get pods -n capv-system
```

The output resembles the following:

```
NAME                                        READY   STATUS    RESTARTS   AGE
capv-controller-manager-785c5978f-nnfns     1/1     Running   0          13h
```

6.16.5.4.1.1 Next Step

vSphere Create a New Air-gapped Cluster (see page 1410)

6.16.5.5 vSphere Create a New Air-gapped Cluster

6.16.5.5.1 Prerequisites

Before you begin, be sure that you have created a Bootstrap Cluster (see page 1409)

6.16.5.5.1.1 Name your Cluster

1. Give your cluster a unique name suitable for your environment. The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[1009] for more naming information.
2. When specifying the `cluster-name`, you must use the same `cluster-name` as used when defining your inventory objects.
3. Set the CLUSTER_NAME environment variable with the command:

```
export CLUSTER_NAME=<my-vsphere-cluster>
```

IMPORTANT: Ensure your subnets (see page 1065) do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

1009 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

- ```
  spec:
    clusterNetwork:
      pods:
        cidrBlocks:
        - 192.168.0.0/16
      services:
        cidrBlocks:
        - 10.96.0.0/12
  ```

6.16.5.5.1.2 Create a New vSphere Kubernetes Cluster

Use the following steps to create a new, air-gapped vSphere cluster.

IMPORTANT: The image must be created by the konvoy-image-builder[1010] project in order to use the registry mirror feature.

1. Configure your cluster to use an existing registry as a mirror when attempting to pull images as done on the vSphere Air-gapped Seed the Registry (see page 1408) page previously.
2. Create the Kubernetes cluster objects by copying the following command and substituting the valid values for your environment:

```
dkp create cluster vsphere \
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <CONTROL_PLANE_IP> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file </path/to/key.pub> \
  --resource-pool <RESOURCE_POOL_NAME> \
  --vm-template konvoy-ova-vsphere-os-release-k8s_release-vsphere-timestamp \
  --virtual-ip-interface eth0 \
  --extra-sans "127.0.0.1" \
  --registry-mirror-url=${REGISTRY_URL} \
  --registry-mirror-cacert=${REGISTRY_CA} \
  --registry-mirror-username=${REGISTRY_USERNAME} \
  --registry-mirror-password=${REGISTRY_PASSWORD} \
  --dry-run \
  --output=yaml \
> ${CLUSTER_NAME}.yaml
```

1010 https://github.com/mesosphere/konvoy-image-builder
```

Expand the drop-downs for **HTTP, FIPS and other flags** to use during the cluster creation step above.

Flatcar

Flatcar OS (see page 1375) use this flag used to instruct the bootstrap cluster to make some changes related to the installation paths:

```
--os-hint flatcar
```

FIPS Requirements

To create a cluster in FIPS mode (see page 1598), inform the controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes by adding those flags to `dkp create cluster` command:

```
--kubernetes-version=v1.28.7+fips.0 \
--etcd-version=3.5.10+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
```

HTTP ONLY

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

```
--http-proxy <<http proxy list>>
--https-proxy <<https proxy list>>
--no-proxy <<no proxy list>>
```

- To configure the Control Plane and Worker nodes to use an HTTP proxy:

```
export CONTROL_PLANE_HTTP_PROXY=http://example.org:8080
export CONTROL_PLANE_HTTPS_PROXY=http://example.org:8080
export
CONTROL_PLANE_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1
,10.96.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.s
vc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.sv
c.cluster,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"

export WORKER_HTTP_PROXY=http://example.org:8080
export WORKER_HTTPS_PROXY=http://example.org:8080
export
WORKER_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96.
0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kube
```

```
rnetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.clust
er,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"
```

- Replace:
  - `example.org,example.com,example.net` with you internal addresses
  - `localhost` and `127.0.0.1` addresses should not use the proxy
  - `10.96.0.0/12` is the default Kubernetes service subnet
  - `192.168.0.0/16` is the default Kubernetes pod subnet
  - `kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.defa ult.svc.cluster,kubernetes.default.svc.cluster.local` is the internal Kubernetes kube-apiserver service
  - `.svc,.svc.cluster,.svc.cluster.local` is the internal Kubernetes services
  - `169.254.169.254` is the AWS metadata server
  - `.elb.amazonaws.com` is for the worker nodes to allow them to communicate directly to the kube-apiserver ELB
- Create a Kubernetes cluster with HTTP proxy configured using the flags and exported variables during `dkp create cluster` command:

```
--control-plane-http-proxy="${CONTROL_PLANE_HTTP_PROXY}" \
--control-plane-https-proxy="${CONTROL_PLANE_HTTPS_PROXY}" \
--control-plane-no-proxy="${CONTROL_PLANE_NO_PROXY}" \
--worker-http-proxy="${WORKER_HTTP_PROXY}" \
--worker-https-proxy="${WORKER_HTTPS_PROXY}" \
--worker-no-proxy="${WORKER_NO_PROXY}" \
```

Individual manifests using the Output Directory flag:

You can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag. This will create multiple files in the specified directory which must already exist:

```
--output-directory=<existing-directory>
```

Refer to the Cluster Creation Customization Choices (see page 1170) section for more information on how to use optional flags such as the `--output-directory` flag.

DKP uses the vsphere CSI driver as the default storage provider (see page 110). Use a Kubernetes CSI[1011] compatible storage that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[1012] for more information.
If you're not using the default, you cannot deploy an alternate provider until ***after*** the `dkp create cluster` is finished. However, it must be determined before Kommander installation.

3. Inspect the created cluster resources with the command:

```
kubectl get clusters,kubeadmcontrolplanes,machinedeployments
```

4. **(Optional)** Edit the cluster objects and familiarize yourself with Cluster API before editing the cluster objects as edits can prevent the cluster from deploying successfully. See Customizing CAPI Components for a Cluster (see page 1061) .
5. Create the cluster from the objects generated in the `dry run`. A warning will appear in the console if the resource already exists and will require you to remove the resource or update your YAML.

```
kubectl create -f ${CLUSTER_NAME}.yaml
```

NOTE: If you used the `--output-directory` flag in your `dkp create .. --dry-run` step above, create the cluster from the objects you created by specifying the directory:

```
kubectl create -f <existing-directory>/
```

6. Use the `wait` command to monitor the cluster control-plane readiness:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=20m
```

Output:

```
cluster.cluster.x-k8s.io/${CLUSTER_NAME} condition met
```

The `READY` status becomes `True` after the cluster control-plane becomes Ready in one of the following steps.

1011 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types
1012 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

After DKP creates the objects on the API server, the Cluster API controllers reconcile them, creating infrastructure and machines. As the controllers progress, they update the Status of each object.

7. Run the DKP `describe` command to monitor the current status of the cluster:

```
dkp describe cluster -c ${CLUSTER_NAME}
```

OUTPUT:

```
NAME                                                          READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/e2e-airgapped-1                                       True
13h
├─ClusterInfrastructure - VSphereCluster/e2e-airgapped-1      True
13h
├─ControlPlane - KubeadmControlPlane/e2e-airgapped-1-control-plane  True
13h
| ├─Machine/e2e-airgapped-1-control-plane-7llgd               True
13h
| ├─Machine/e2e-airgapped-1-control-plane-vncbl               True
13h
| └─Machine/e2e-airgapped-1-control-plane-wbgrm               True
13h
└─Workers
    └─MachineDeployment/e2e-airgapped-1-md-0                    True
13h
    ├─Machine/e2e-airgapped-1-md-0-74c849dc8c-67rv4           True
13h
    ├─Machine/e2e-airgapped-1-md-0-74c849dc8c-n2skc           True
13h
    ├─Machine/e2e-airgapped-1-md-0-74c849dc8c-nkftv           True
13h
    └─Machine/e2e-airgapped-1-md-0-74c849dc8c-sqklv           True
13h
```

6. As they progress, the controllers also create Events, which you can list using the command:

```
kubectl get events | grep ${CLUSTER_NAME}
```

For brevity, this example uses `grep`. You can also use separate commands to get Events for specific objects, such as `kubectl get events --field-selector involvedObject.kind="VSphereCluster"` and `kubectl get events --field-selector involvedObject.kind="VSphereMachine"`.

6.16.5.5.1.3 Known Limitations

NOTE: Be aware of these limitations in the current release of DKP Konvoy.

- The DKP Konvoy version used to create a bootstrap cluster must match the DKP Konvoy version used to create a workload cluster.
- DKP Konvoy supports deploying one workload cluster.
- DKP Konvoy generates a set of objects for one Node Pool.
- DKP Konvoy does not validate edits to cluster objects.

6.16.5.5.1.4 Next Step

vSphere Make your Air-gapped Cluster Self-Managed (see page 1416)

6.16.5.6 vSphere Make your Air-gapped Cluster Self-Managed

Konvoy deploys all cluster lifecycle services to a bootstrap cluster, which then deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, which makes the workload cluster self-managed. This section describes how to make a workload cluster self-managed.

Before starting, ensure you create a workload cluster as described in Create a New vSphere Cluster or Create a New Air-gapped vSphere Cluster.

This page contains instructions on how to make your cluster self-managed. This is necessary if there is only one cluster in your environment, or if this cluster should become the Management cluster in a multi-cluster environment.

If you already have a self-managed or Management cluster in your environment, skip this page.

6.16.5.6.1 Make the New Kubernetes Cluster Manage Itself

To create a Management Cluster or stand alone Essential Cluster, the cluster must manage itself. Follow these steps to achieve:

1. Deploy cluster lifecycle services on the workload cluster:

```
dkp create capi-components --kubeconfig ${CLUSTER_NAME}.conf
```

Output:

```
✓ Initializing new CAPI components
```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

2. Move the Cluster API objects from the bootstrap to the workload cluster:
   The cluster lifecycle services on the workload cluster are ready, but the workload cluster configuration is on the bootstrap cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the bootstrap to the workload cluster. This process is also called a Pivot[1013].

   ```
   dkp move capi-resources --to-kubeconfig ${CLUSTER_NAME}.conf
   ```

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

Output:

```
 ✓ Moving cluster resources
You can now view resources in the moved cluster by using the --kubeconfig flag with
kubectl. For example: kubectl --kubeconfig=vsphere-example.conf get nodes
```

1013 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

To ensure only one set of cluster lifecycle services manages the workload cluster, Konvoy first pauses reconciliation of the objects on the bootstrap cluster, then creates the objects on the workload cluster. As Konvoy copies the objects, the cluster lifecycle services on the workload cluster reconcile the objects. The workload cluster becomes self-managed after Konvoy creates all the objects. If it fails, the move command can be safely retried.

3. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf wait --for=condition=ControlPlaneRead
y "clusters/${CLUSTER_NAME}" --timeout=20m
```

Output:

```
cluster.cluster.x-k8s.io/vsphere-example condition met
```

After moving the cluster lifecycle services to the workload cluster, remember to use Konvoy with the workload cluster kubeconfig.

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

```
dkp describe cluster --kubeconfig ${CLUSTER_NAME}.conf -c ${CLUSTER_NAME}
```

Output:

```
NAME                                                           READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/vsphere-example-1                                      True
13h
├─ClusterInfrastructure - VSphereCluster/vsphere-example-1     True
13h
├─ControlPlane - KubeadmControlPlane/vsphere-example-control-plane  True
13h
│ ├─Machine/vsphere-example-control-plane-7llgd                True
13h
│ ├─Machine/vsphere-example-control-plane-vncbl                True
13h
```

```
│ └─Machine/vsphere-example-control-plane-wbgrm                    True
13h
└─Workers
    └─MachineDeployment/vsphere-example-md-0                     True
13h
    ├─Machine/vsphere-example-md-0-74c849dc8c-67rv4              True
13h
    ├─Machine/vsphere-example-md-0-74c849dc8c-n2skc              True
13h
    ├─Machine/vsphere-example-md-0-74c849dc8c-nkftv              True
13h
    └─Machine/vsphere-example-md-0-74c849dc8c-sqklv              True
13h
```

5. Remove the bootstrap cluster, if desired, as the workload cluster is now self-managed:

```
dkp delete bootstrap
```

Output:

```
✓ Deleting bootstrap cluster
```

6.16.5.6.1.1 Known Limitations

Be aware of these limitations in the current release of DKP Konvoy.

- Before making a workload cluster self-managed, be sure that its control plane nodes have sufficient permissions for running Cluster API controllers.
- DKP Konvoy supports moving only one set of cluster objects from the bootstrap cluster to the workload cluster, or vice-versa.
- DKP Konvoy only supports moving all namespaces in the cluster; DKP does not support migration of individual namespaces.

6.16.5.6.1.2 Next Step

vSphere Explore your Air-gapped Cluster (see page 1419)

6.16.5.7 vSphere Explore your Air-gapped Cluster

6.16.5.7.1 Find and Change Configurations

Your newly created cluster has many components to explore that are described below.

6.16.5.7.1.1 Get the `kubeconfig` File for the New Kubernetes Cluster

1. Fetch the kubeconfig file with the command:

```
dkp get kubeconfig -c ${air-gapped_NAME} > ${air-gapped_NAME}.conf
```

6.16.5.7.1.2 Create a StorageClass with a vSphere Datastore

Follow these steps:

1. Access the Datastore tab in the vSphere client and select a datastore by name.
2. Copy the URL of that datastore from the information dialog that displays.
3. Return to the DKP CLI, and delete the existing `StorageClass` with the command:

```
kubectl delete storageclass vsphere-raw-block-sc
```

4. Run the following command to create a new StorageClass, supplying the correct values for your environment:

```
cat <<EOF > vsphere-raw-block-sc.yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  name: vsphere-raw-block-sc
provisioner: csi.vsphere.vmware.com
parameters:
  datastoreurl: "<url>"
volumeBindingMode: WaitForFirstConsumer
EOF
```

6.16.5.7.1.3 Explore Nodes and Pods in the New Cluster

1. List the Nodes with this command:

```
kubectl --kubeconfig=${air-gapped_NAME}.conf get nodes
```

NOTE: It may take a few minutes for the Status to move to Ready while the Pod network is deployed. The Node's Status should change to Ready soon after the calico-node DaemonSet Pods are Ready.

The output resembles this example:

```
NAME                                          STATUS   ROLES                  AGE
VERSION
d2iq-e2e-air-gapped-1-control-plane-7llgd     Ready    control-plane,master   20h
v1.28.7
d2iq-e2e-air-gapped-1-control-plane-vncbl     Ready    control-plane,master   19h
v1.28.7
d2iq-e2e-air-gapped-1-control-plane-wbgrm     Ready    control-plane,master   19h
v1.28.7
d2iq-e2e-air-gapped-1-md-0-74c849dc8c-67rv4   Ready    <none>                 19h
v1.28.7
d2iq-e2e-air-gapped-1-md-0-74c849dc8c-n2skc   Ready    <none>                 19h
v1.28.7
d2iq-e2e-air-gapped-1-md-0-74c849dc8c-nkftv   Ready    <none>                 19h
v1.28.7
d2iq-e2e-air-gapped-1-md-0-74c849dc8c-sqklv   Ready    <none>                 19h
v1.28.7
```

2. List the pods with the command:

```
kubectl --kubeconfig=${air-gapped_NAME}.conf get pods -A
```

The output resembles the following example:

```
NAMESPACE              NAME
READY   STATUS     RESTARTS      AGE
calico-system          calico-kube-controllers-57fbd7bd59-qqd96
1/1     Running    0             20h
calico-system          calico-node-2m524
1/1     Running    3 (19h ago)   19h
calico-system          calico-node-bbhg5
1/1     Running    0             20h
calico-system          calico-node-cc5lf
1/1     Running    2 (19h ago)   19h
calico-system          calico-node-cwg7x
1/1     Running    1 (19h ago)   19h
calico-system          calico-node-d59hn
1/1     Running    1 (19h ago)   19h
calico-system          calico-node-qmmcz
1/1     Running    0             19h
```

```
calico-system            calico-node-wdqhx
1/1     Running    0             19h
calico-system            calico-typha-655489d8cc-b5jnt
1/1     Running    0             20h
calico-system            calico-typha-655489d8cc-q92x9
1/1     Running    0             19h
calico-system            calico-typha-655489d8cc-vjlkx
1/1     Running    0             19h
kube-system              cluster-autoscaler-68c759fbf6-7d2ck
0/1     Init:0/1   0             20h
kube-system              coredns-78fcd69978-qn4qt
1/1     Running    0             20h
kube-system              coredns-78fcd69978-wqpmg
1/1     Running    0             20h
kube-system              etcd-d2iq-e2e-air-gapped-1-control-plane-7llgd
1/1     Running    0             20h
kube-system              etcd-d2iq-e2e-air-gapped-1-control-plane-vncbl
1/1     Running    0             19h
kube-system              etcd-d2iq-e2e-air-gapped-1-control-plane-wbgrm
1/1     Running    0             19h
kube-system              kube-apiserver-d2iq-e2e-air-gapped-1-control-plane-7llgd
1/1     Running    0             20h
kube-system              kube-apiserver-d2iq-e2e-air-gapped-1-control-plane-vncbl
1/1     Running    0             19h
kube-system              kube-apiserver-d2iq-e2e-air-gapped-1-control-plane-wbgrm
1/1     Running    0             19h
kube-system              kube-controller-manager-d2iq-e2e-air-gapped-1-control-
plane-7llgd   1/1     Running    1 (19h ago)   20h
kube-system              kube-controller-manager-d2iq-e2e-air-gapped-1-control-plane-
vncbl   1/1     Running    0             19h
kube-system              kube-controller-manager-d2iq-e2e-air-gapped-1-control-plane-
wbgrm   1/1     Running    0             19h
kube-system              kube-proxy-cpscs
1/1     Running    0             19h
kube-system              kube-proxy-hhmxq
1/1     Running    0             19h
kube-system              kube-proxy-hxhnk
1/1     Running    0             19h
kube-system              kube-proxy-nsrbp
1/1     Running    0             19h
kube-system              kube-proxy-scxfg
1/1     Running    0             20h
kube-system              kube-proxy-tth4k
1/1     Running    0             19h
kube-system              kube-proxy-x2xfx
1/1     Running    0             19h
kube-system              kube-scheduler-d2iq-e2e-air-gapped-1-control-plane-7llgd
1/1     Running    1 (19h ago)   20h
kube-system              kube-scheduler-d2iq-e2e-air-gapped-1-control-plane-vncbl
1/1     Running    0             19h
kube-system              kube-scheduler-d2iq-e2e-air-gapped-1-control-plane-wbgrm
1/1     Running    0             19h
```

```
kube-system              kube-vip-d2iq-e2e-air-gapped-1-control-plane-7llgd
1/1     Running    1 (19h ago)   20h
kube-system              kube-vip-d2iq-e2e-air-gapped-1-control-plane-vncbl
1/1     Running    0             19h
kube-system              kube-vip-d2iq-e2e-air-gapped-1-control-plane-wbgrm
1/1     Running    0             19h
kube-system              vsphere-cloud-controller-manager-4zj7q
1/1     Running    0             19h
kube-system              vsphere-cloud-controller-manager-87tgm
1/1     Running    0             19h
kube-system              vsphere-cloud-controller-manager-xqmn4
1/1     Running    1 (19h ago)   20h
node-feature-discovery   node-feature-discovery-master-84c67dcbb6-txfw9
1/1     Running    0             20h
node-feature-discovery   node-feature-discovery-worker-8tg2l
1/1     Running    3 (19h ago)   19h
node-feature-discovery   node-feature-discovery-worker-c5f6q
1/1     Running    0             19h
node-feature-discovery   node-feature-discovery-worker-fjfkm
1/1     Running    0             19h
node-feature-discovery   node-feature-discovery-worker-x6tz8
1/1     Running    0             19h
tigera-operator          tigera-operator-d499f5c8f-r2srj
1/1     Running    1 (19h ago)   20h
vmware-system-csi        vsphere-csi-controller-7ffd6884cc-d7rql
7/7     Running    5 (19h ago)   20h
vmware-system-csi        vsphere-csi-controller-7ffd6884cc-k82cm
7/7     Running    2 (19h ago)   20h
vmware-system-csi        vsphere-csi-controller-7ffd6884cc-qttkp
7/7     Running    1 (19h ago)   20h
vmware-system-csi        vsphere-csi-node-678hw
3/3     Running    0             19h
vmware-system-csi        vsphere-csi-node-6tbsh
3/3     Running    0             19h
vmware-system-csi        vsphere-csi-node-9htwr
3/3     Running    5 (20h ago)   20h
vmware-system-csi        vsphere-csi-node-g8r6l
3/3     Running    0             19h
vmware-system-csi        vsphere-csi-node-ghmr6
3/3     Running    0             19h
vmware-system-csi        vsphere-csi-node-jhvgm
3/3     Running    0             19h
vmware-system-csi        vsphere-csi-node-rp77r
3/3     Running    0             19h
```

6.16.5.7.1.4 Next Step

vSphere Install Kommander in an Air-gapped Environment (see page 1424)

6.16.5.8 vSphere Install Kommander in an Air-gapped Environment

This section provides installation instructions for the Kommander component of DKP in an air-gapped vSphere environment.

6.16.5.8.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry: Air-gapped Environments (see page 1536).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

6.16.5.8.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

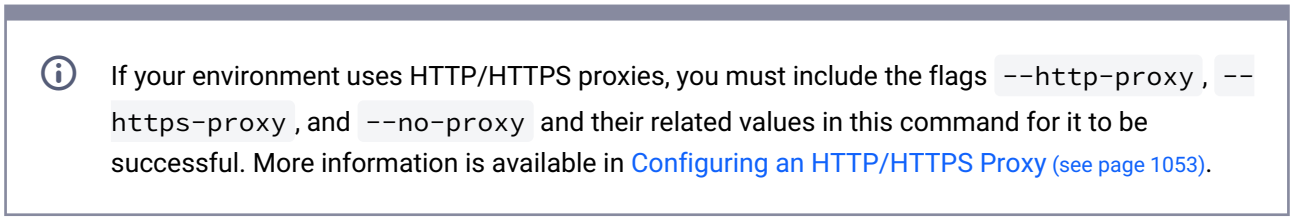
3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init --airgapped > kommander.yaml
   ```

4. *If required*, **customize** your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include:
   - Custom Domains and Certificates
   - HTTP proxy
   - External Load Balancer
   - GPU utilization, etc.
   - Rook Ceph customization for Pre-provisioned environments (see page 1541)
5. *If required*: If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
...
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true
...
```

6. Expand **one** of the following sets of instructions, depending on your license and application environments:



If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

Essential License: Install Kommander in an Air-Gapped Environment

6.16.5.8.1.2 Essential License: Install Kommander in an Air-gapped Environment

Use the customized `kommander.yaml` to install DKP in an air-gapped environment:

```
dkp install kommander \
--installer-config kommander.yaml --kubeconfig=${CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.0.tar.gz
```

Enterprise License: Install Kommander in an Air-gapped Environment with DKP Catalog Applications

6.16.5.8.1.3 Enterprise License: Install Kommander in an Air-gapped Environment with DKP Catalog Applications

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add the following values to enable DKP Catalog Applications:

```
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./dkp-catalog-applications-v2.8.0.tar.gz
...
```

⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander \
--installer-config kommander.yaml --kubeconfig=${CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.0.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.8.0.tar.gz
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

6.16.5.8.2 Kommander Customizations

You can configure the Kommander component of DKP during the initial installation, and also post-installation using the DKP CLI. If you are not sure of what you want to customize during install, then proceed to the next step. To read about Kommander component customization options, refer to this section of the documetation: Kommander Customizations (see page 1558)

6.16.5.8.3 Next Step

Day 2 - Cluster Operations Management (see page 590)

6.16.6 vSphere Management Tools

After cluster creation and configuration, you can revisit clusters to update and change variables.

- Manage vSphere Node Pools (see page 1427)
- vSphere Certificate Renewal (see page 1435)
- vSphere Cluster Autoscaler (see page 1437)
- Delete vSphere Cluster (see page 1440)

6.16.6.1 Manage vSphere Node Pools

Node pools are part of a cluster and managed as a group, and you can use a node pool to manage a group of machines using the same common properties. When Konvoy creates a new default cluster, there is one node pool for the worker nodes, and all nodes in that new node pool have the same configuration.

You can create additional node pools for more specialized hardware or configuration. For example, if you want to tune your memory usage on a cluster where you need maximum memory for some machines and minimal memory on other machines, you would create a new node pool with those specific resource needs.

Konvoy implements node pools using Cluster API MachineDeployments[1014].

6.16.6.1.1 Create vSphere Node Pools

Creating a node pool is useful when you need to run workloads that require machines with specific resources, such as a GPU, additional memory, or specialized network or storage hardware.

6.16.6.1.1.1 Prepare the environment

Follow these steps:

1. Set the environment variable to the name you assigned this cluster with the command:

   ```
   export CLUSTER_NAME=my-vsphere-cluster
   ```

2. If your workload cluster is self-managed, as described in Make the New Cluster Self-Managed (see page 1394), configure `kubectl` to use the kubeconfig for the cluster:

1014 https://cluster-api.sigs.k8s.io/developer/architecture/controllers/machine-deployment.html

```
export KUBECONFIG=${CLUSTER_NAME}.conf
```

3. Define your node pool name:

```
export NODEPOOL_NAME=example
```

6.16.6.1.1.2 Create a vSphere node pool

Create a new vSphere node pool with 3 replicas using this command:

```
dkp create nodepool vsphere ${NODEPOOL_NAME} \
  --cluster-name=${CLUSTER_NAME} \
  --network=example_network \
  --data-center=example_datacenter \
  --data-store=example_datastore \
  --folder=example_folder \
  --server=example_vsphere_api_server_url\
  --resource-pool=example_resource_pool \
  --vm-template=example_vm_template \
  --replicas=3
```

The output resembles this example:

```
machinedeployment.cluster.x-k8s.io/example created
vspheremachinetemplate.infrastructure.cluster.x-k8s.io/example created
kubeadmconfigtemplate.bootstrap.cluster.x-k8s.io/example created
 ✓ Creating default/example nodepool resources
```

This example uses default values for brevity. Advanced users can use a combination of the `--dry-run` and `--output=yaml` or `--output-directory=<your-target-directory>/` flags to get a complete set of node pool objects to modify locally or store in version control.

6.16.6.1.2 List vSphere Node Pools

Use this command to list the node pools of a given cluster. This returns specific properties of each node pool so that you can see the name of the MachineDeployments.

To list all node pools for a managed cluster, run the command:

```
dkp get nodepools --cluster-name=${CLUSTER_NAME} --kubeconfig=${CLUSTER_NAME}.conf
```

The expected output is similar to the following example, indicating the desired size of the node pool, the number of replicas ready in the node pool, and the Kubernetes version those nodes are running:

```
NODEPOOL                    DESIRED             READY               KUBERNETES
VERSION
demo-cluster-md-0           4                   4                   v1.27.6
example                     3                   0                   v1.27.6
```

6.16.6.1.3 Scale vSphere Node Pools

While you can run Cluster Autoscaler (see page 1437), you can also manually scale your node pools up or down when you need more finite control over your environment. For example, if you require 10 machines to run a process, you can manually set the scaling to run those 10 machines only. However, if also using the Cluster Autoscaler, you must stay within your minimum and maximum bounds.

6.16.6.1.3.1 Scaling up Node Pools

1. To scale up a node pool in a cluster, run the command that follows, replacing the value 5 with the actual number of replicas you need:

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=5 --cluster-name=${CLUSTER_NAME}
```

Your output should be similar to this example, indicating the scaling is in progress:

```
INFO[2021-07-26T08:54:35-07:00] Running scale nodepool command
clusterName=demo-cluster managementClusterKubeconfig= namespace=default src="nodepool
/scale.go:82"
INFO[2021-07-26T08:54:35-07:00] Nodepool example scaled to 5 replicas
clusterName=demo-cluster managementClusterKubeconfig= namespace=default src="nodepool
/scale.go:94"
```

2. After a few minutes, you can list the node pools with the command:

```
dkp get nodepools --cluster-name=${CLUSTER_NAME} --kubeconfig=${CLUSTER_NAME}.conf
```

Your output should be similar to this example, with the number of DESIRED and READY replicas increased to 5:

```
NODEPOOL                    DESIRED             READY               KUBERNETES
VERSION
example                     5                   5                   v1.28.7
demo-cluster-md-0           4                   4                   v1.28.7
```

6.16.6.1.3.2 Scaling Down Node Pools

1. To scale down a node pool, run the command:

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=4 --cluster-name=${CLUSTER_NAME}
```

Output:

```
INFO[2021-07-26T08:54:35-07:00] Running scale nodepool command
clusterName=demo-cluster managementClusterKubeconfig= namespace=default src="nodepool
/scale.go:82"
INFO[2021-07-26T08:54:35-07:00] Nodepool example scaled to 4 replicas
clusterName=demo-cluster managementClusterKubeconfig= namespace=default src="nodepool
/scale.go:94"
```

In a default cluster, the nodes to delete are selected at random. This behavior is controller by CAPI's delete policy[1015]. However, when using the Konvoy CLI to scale down a node pool, you can specify the Kubernetes Nodes you want to delete.

To do this, set the flag `--nodes-to-delete` with a list of nodes as shown in the next command. This adds an annotation `cluster.x-k8s.io/delete-machine=yes` to the matching Machine object that contains `status.NodeRef` with the node names from `--nodes-to-delete`.

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=3 --nodes-to-delete=<> --cluster-
name=${CLUSTER_NAME}
```

Output:

```
INFO[2021-07-26T08:54:35-07:00] Running scale nodepool command
clusterName=demo-cluster managementClusterKubeconfig= namespace=default src="nodepool
/scale.go:82"
INFO[2021-07-26T08:54:35-07:00] Nodepool example scaled to 3 replicas
clusterName=demo-cluster managementClusterKubeconfig= namespace=default src="nodepool
/scale.go:94"
```

1015 https://github.com/kubernetes-sigs/cluster-api/blob/v0.4.0/api/v1alpha4/machineset_types.go#L85-L105

6.16.6.1.3.3 Scaling Node Pools when Using Cluster Autoscaler

If you configured the cluster autoscaler[1016] for the `demo-cluster-md-0` node pool, the value of `--replicas` must be within the minimum and maximum bounds.

1. For example, assuming you have the these annotations:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment ${NODEPOOL_NAME}
cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size=2
kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment ${NODEPOOL_NAME}
cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size=6
```

2. Try to scale the node pool to 7 replicas with the command:

```
dkp scale nodepools ${NODEPOOL_NAME} --replicas=7 -c demo-cluster
```

3. This action results in an error similar to:

```
INFO[2021-07-26T09:46:37-07:00] Running scale nodepool command
clusterName=demo-cluster managementClusterKubeconfig= namespace=default src="nodepool
/scale.go:82"
Error: failed to scale nodepool: scaling MachineDeployment is forbidden: desired
replicas 7 is greater than the configured max size annotation cluster.x-k8s.io/
cluster-api-autoscaler-node-group-max-size: 6
```

Similarly, scaling down to a number of replicas less than the configured `min-size` also returns an error.

6.16.6.1.4 Replace a vSphere Node

6.16.6.1.4.1 Prerequisites

Before you begin, you must:

- Create a workload cluster (see page 1386) or create an air-gapped workload cluster (see page 1410).
- Make the new cluster self-managed (see page 1394).

1016 https://docs.d2iq.com/dkp/konvoy/2.2/choose-infrastructure/vsphere/nodepools/cluster-autoscaler

Replace a Worker Node

In certain situations, you may want to delete a worker node and have Cluster API[1017] replace it with a newly-provisioned machine.

1. Identify the name of the node to delete.

   List the nodes:

   ```
   kubectl --kubeconfig ${CLUSTER_NAME}.conf get nodes
   ```

   The output from this command resembles the following:

   ```
   NAME                                          STATUS   ROLES                  AGE
   VERSION
   d2iq-e2e-cluster-1-control-plane-7llgd        Ready    control-plane,master   20h
   v1.28.7
   d2iq-e2e-cluster-1-control-plane-vncbl        Ready    control-plane,master   20h
   v1.28.7
   d2iq-e2e-cluster-1-control-plane-wbgrm        Ready    control-plane,master   19h
   v1.28.7
   d2iq-e2e-cluster-1-md-0-74c849dc8c-67rv4      Ready    <none>                 20h
   v1.28.7
   d2iq-e2e-cluster-1-md-0-74c849dc8c-n2skc      Ready    <none>                 20h
   v1.28.7
   d2iq-e2e-cluster-1-md-0-74c849dc8c-nkftv      Ready    <none>                 20h
   v1.28.7
   d2iq-e2e-cluster-1-md-0-74c849dc8c-sqklv      Ready    <none>                 20h
   v1.28.7
   ```

2. Export a variable with the node name to use in the next steps:

   This example uses the name `d2iq-e2e-cluster-1-md-0-74c849dc8c-67rv4`.

   ```
   export NAME_NODE_TO_DELETE="d2iq-e2e-cluster-1-md-0-74c849dc8c-67rv4"
   ```

3. Delete the Machine resource with the command:

1017 https://cluster-api.sigs.k8s.io/

```
NAME_MACHINE_TO_DELETE=$(kubectl --kubeconfig ${CLUSTER_NAME}.conf get machine
-ojsonpath="{.items[?
(@.status.nodeRef.name==\"$NAME_NODE_TO_DELETE\")].metadata.name}")
kubectl --kubeconfig ${CLUSTER_NAME}.conf delete machine
"$NAME_MACHINE_TO_DELETE"
```

Output:

```
machine.cluster.x-k8s.io "d2iq-e2e-cluster-1-md-0-74c849dc8c-67rv4" deleted
```

The command does not return immediately, but it does return after the Machine resource is deleted.

A few minutes after the Machine resource is deleted, the corresponding Node resource is also deleted.

4. Observe the Machine resource replacement using this command:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf get machinedeployment
```

Output:

```
NAME                      CLUSTER              REPLICAS   READY   UPDATED
UNAVAILABLE   PHASE       AGE   VERSION
d2iq-e2e-cluster-1-md-0   d2iq-e2e-cluster-1   4          3       4         1
ScalingUp   20h   1.28.7
```

In this example, there exist 4 replicas, but only 3 are ready. One replica is unavailable, and the `ScalingUp` phase means a new Machine is being created.

5. Identify the replacement Machine using this command:

```
export NAME_NEW_MACHINE=$(kubectl --kubeconfig ${CLUSTER_NAME}.conf get
machines \
    -l=cluster.x-k8s.io/deployment-name=${CLUSTER_NAME}-md-0 \
    -ojsonpath='{.items[?(@.status.phase=="Provisioning")].metadata.name}
{"\n"}')
echo "$NAME_NEW_MACHINE"
```

If the output is empty, the new Machine has probably exited the `Provisioning` phase and entered the `Running` phase.

6. Identify the replacement Node using this command:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf get nodes \
    -o=jsonpath="{.items[?(@.metadata.annotations.cluster\.x-k8s\.io/
machine==\"$NAME_NEW_MACHINE\")].metadata.name}"
```

The output should be similar to this example:

```
d2iq-e2e-cluster-1-md-0-74c849dc8c-rc528
```

If the output is empty, the Node resource is not yet available, or does not yet have the expected annotation. Wait a few minutes, then repeat the command.

6.16.6.1.5 Delete vSphere Node Pools

Deleting a node pool deletes the Kubernetes nodes and the underlying infrastructure. DKP drains all nodes prior to deletion and reschedules the pods running on those nodes.

To delete a node pool from a managed cluster, run the command:

```
dkp delete nodepool ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME}
```

Here, `example` is the node pool to be deleted.

The expected output is similar to the following example, indicating the node pool is being deleted:

```
INFO[2021-07-28T17:14:26-07:00] Running nodepool delete command
Nodepool=example clusterName=d2iq-e2e-cluster-1 managementClusterKubeconfig=
namespace=default src="nodepool/delete.go:80"
```

Deleting an invalid node pool results in output similar to this example command output:

```
dkp delete nodepool ${CLUSTER_NAME}-md-invalid --cluster-name=${CLUSTER_NAME}

INFO[2021-07-28T17:11:44-07:00] Running nodepool delete command
Nodepool=demo-cluster-md-invalid clusterName=d2iq-e2e-cluster-1
 managementClusterKubeconfig= namespace=default src="nodepool/delete.go:80"
Error: failed to get nodepool with name demo-cluster-md-invalid in namespace default
: failed to get nodepool with name demo-cluster-md-invalid in namespace default :
machinedeployments.cluster.x-k8s.io "demo-cluster-md-invalid" not found
```

6.16.6.2 vSphere Certificate Renewal

6.16.6.2.1 Configure Automated Renewal for Managed Kubernetes PKI Certficates

6.16.6.2.2 Certificate Renewal

During cluster creation, Kubernetes establishes a Public Key Infrastructure (PKI) for generating the TLS certificates needed for securing cluster communication for various components such as `etcd`, `kubernetes-apiserver` and `kube-proxy`. The certificates created by these components have a default expiration of one year and are renewed when an administrator updates the cluster.

Kubernetes provides a facility to renew all certificates automatically during control plane updates. For administrators who need long-running clusters or clusters that are not upgraded often, `dkp` provides automated certificate renewal, without a cluster upgrade.

6.16.6.2.2.1 Requirements

This feature requires that you install Python 3.5 or higher version on all control plane hosts.

6.16.6.2.3 Prerequisites

- Complete the bootstrap Cluster Lifecycle (see page 1384) topic.

6.16.6.2.3.1 Create a cluster with automated certificate renewal

To enable the automated certificate renewal, create a Konvoy cluster using the `certificate-renew-interval` flag:

```
dkp create cluster vsphere --certificate-renew-interval=60 --cluster-name=long-
running
```

The `certificate-renew-interval` is the number of days after which Kubernetes-managed PKI certificates will be renewed. For example, an `certificate-renew-interval` value of 30 means the certificates are renewed every 30 days.

6.16.6.2.3.2 Technical details

The following manifests are modified on the control plane hosts, and are located at `/etc/kubernetes/manifests`. Modifications to these files requires SUDO access.

```
kube-controller-manager.yaml
kube-apiserver.yaml
kube-scheduler.yaml
kube-proxy.yaml
```

The following annotation indicates the time each component was reset:

```
metadata:
  annotations:
    konvoy.d2iq.io/restartedAt: $(date +%s)
```

This only occurs when the PKI certificates are older than the interval given at cluster creation time. This is activated by a `systemd` `timer` called `renew-certs.timer` that triggers an associated `systemd` service called `renew-certs.service` that runs on all of the control plane hosts.

6.16.6.2.3.3 Debugging

To debug the automatic certificate renewal feature, a cluster administrator can look at several different components to see if the certificates were renewed. For example, an administrator might start with a look at the control plane pod definition to check the last reset time. To determine if a scheduler pod was properly reset, run the command:

```
kubectl get pod -n kube-system kube-scheduler-nodename -o yaml
```

The output of the command is similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    konvoy.d2iq.io/restartedAt: "1626124940.735733"
```

Administrators who want more details on the execution of the `systemd` service can use `ssh` to connect to the control plane hosts, and then use the `systemctl` and `journalctl` commands that follow to help diagnose potential issues.

Check timer status

To check the status of the timers, when they last ran, and when they are scheduled to run next, use the command:

```
systemctl list-timers
```

Check renew certs status

To check the status of the `renew-certs` service, use the command:

```
systemctl status renew-certs
```

Review logs

To get the logs of the last run of the service, use the command:

```
journalctl logs -u renew-certs
```

6.16.6.3 vSphere Cluster Autoscaler

6.16.6.3.1 Cluster Autoscaler

Cluster Autoscaler[1018] provides the ability to automatically scale-up or scale-down the number of worker nodes in a cluster, based on the number of pending pods to be scheduled. Running the Cluster Autoscaler is optional.

Unlike Horizontal-Pod Autoscaler[1019], Cluster Autoscaler does not depend on any Metrics server and does not need Prometheus or any other metrics source.

The Cluster Autoscaler looks at the following annotations on a MachineDeployment to determine its scale-up and scale-down ranges:

```
cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size
cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size
```

1018 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi
1019 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-fast-is-hpa-when-combined-with-ca

The full list of command line arguments to the Cluster Autoscaler controller is found this topic[1020].

For more information about how Cluster Autoscaler works, see these documents:

- What is Cluster Autoscaler[1021]
- How does scale-up work[1022]
- How does scale-down work[1023]
- CAPI Provider for Cluster Autoscaler[1024]

6.16.6.3.1.1 Cluster Autoscaler Prerequisites

Before you begin, you must have:

- A Bootstrap Cluster Lifecycle (see page 1384).
- Created a new Kubernetes Cluster (see page 1386).
- A Self-Managed Cluster (see page 1394).

6.16.6.3.1.2 Run Cluster Autoscaler

The Cluster Autoscaler controller runs on the workload cluster. Upon creation of the workload cluster, this controller does not have all the objects required to function correctly until after a `dkp move` is issued from the bootstrap cluster.

Run the following steps to enable Cluster Autoscaler:

1. Ensure the Cluster Autoscaler controller is up and running (no restarts and no errors in the logs)

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf logs deployments/cluster-autoscaler
   cluster-autoscaler -n kube-system -f
   ```

2. Enable Cluster Autoscaler by setting the min & max ranges

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment $
   {NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size=2
   kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment $
   {NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size=6
   ```

3. The Cluster Autoscaler logs will show that the worker nodes are associated with node-groups and that pending pods are being watched.

1020 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-are-the-parameters-to-ca
1021 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-is-cluster-autoscaler
1022 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-up-work
1023 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-down-work
1024 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi

4. To demonstrate that it is working properly, create a large deployment which will trigger pending pods.

```
cat <<EOF | kubectl --kubeconfig=${CLUSTER_NAME}.conf apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox-deployment
  labels:
    app: busybox
spec:
  replicas: 600
  selector:
    matchLabels:
      app: busybox
  template:
    metadata:
      labels:
        app: busybox
    spec:
      containers:
      - name: busybox
        image: busybox:latest
        command:
          - sleep
          - "3600"
        imagePullPolicy: IfNotPresent
      restartPolicy: Always
EOF
```

   Cluster Autoscaler scales up the number of Worker Nodes until there are no pending pods.

5. Scale down the number of replicas for `busybox-deployment` with the command:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf scale --replicas=30 deployment/
busybox-deployment
```

6. Cluster Autoscaler starts to scale down the number of Worker Nodes after the default timeout of 10 minutes.

6.16.6.4 Delete vSphere Cluster

6.16.6.4.1 Prepare to delete a self-managed workload cluster

> A self-managed workload cluster cannot delete itself. If your workload cluster is self-managed, you must create a bootstrap cluster and move the cluster lifecycle services to the bootstrap cluster before deleting the workload cluster.

If you did not make your workload cluster self-managed, as described in Make New Cluster Self-Managed (see page 1394), see Delete the workload cluster (see page 1442).

1. Create a bootstrap cluster:

   The bootstrap cluster will host the Cluster API controllers that reconcile the cluster objects marked for deletion. To avoid using the wrong kubeconfig, the following steps use **explicit** kubeconfig paths and contexts.

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config
   ```

   The output resembles this example:

   ```
   ✓ Creating a bootstrap cluster
   ✓ Initializing new CAPI components
   ```

2. Move the Cluster API objects from the workload to the bootstrap cluster: The cluster lifecycle services on the bootstrap cluster are ready, but the workload cluster configuration is on the workload cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the workload to the bootstrap cluster. This process is also called a Pivot[1025].

   ```
   dkp move \
       --from-kubeconfig ${CLUSTER_NAME}.conf \
       --from-context konvoy-${CLUSTER_NAME}-admin@konvoy-${CLUSTER_NAME} \
       --to-kubeconfig $HOME/.kube/config \
       --to-context kind-konvoy-capi-bootstrapper
   ```

1025 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
INFO[2021-06-09T11:47:11-07:00] Running pivot command
fromClusterKubeconfig=aws-example.conf fromClusterContext= src="move/
move.go:83" toClusterKubeconfig=/home/clusteradmin/.kube/config
toClusterContext=
INFO[2021-06-09T11:47:36-07:00] Pivot operation complete.
src="move/move.go:108"
INFO[2021-06-09T11:47:36-07:00] You can now view resources in the moved cluster
by using the --kubeconfig flag with kubectl. For example: kubectl --
kubeconfig=/home/clusteradmin/.kube/config get nodes  src="move/move.go:155"
```

3. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

```
dkp describe cluster --kubeconfig $HOME/.kube/config -c ${CLUSTER_NAME}
```

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/d2iq-e2e-cluster_name-1                                     True
13h
├─ClusterInfrastructure - VSphereCluster/d2iq-e2e-cluster_name-1    True
13h
├─ControlPlane - KubeadmControlPlane/d2iq-control-plane             True
13h
│ ├─Machine/d2iq--control-plane-7llgd                               True
13h
│ ├─Machine/d2iq--control-plane-vncbl                               True
13h
│ └─Machine/d2iq--control-plane-wbgrm                               True
13h
└─Workers
    └─MachineDeployment/d2iq--md-0                                  True
13h
    ├─Machine/d2iq--md-0-74c849dc8c-67rv4                           True
13h
    ├─Machine/d2iq--md-0-74c849dc8c-n2skc                           True
13h
    ├─Machine/d2iq--md-0-74c849dc8c-nkftv                           True
13h
    └─Machine/d2iq--md-0-74c849dc8c-sqklv                           True
13h
```

After moving the cluster lifecycle services to the workload cluster, remember to use dkp with the workload cluster kubeconfig. Use DKP with the bootstrap cluster to delete the workload cluster.

4. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig $HOME/.kube/config wait --for=condition=controlplaneready
"clusters/${CLUSTER_NAME}" --timeout=60m
```

The output should be similar to this example:

```
d2iq-e2e-cluster-1-control-plane/vsphere-example condition met
```

Persistent Volumes (PVs) are not deleted automatically by design in order to preserve your data. However, they take up storage space if not deleted. You must delete PVs manually. Information for backup of a cluster and PVs is on the page in documentation called Back up your Cluster's Applications and Persistent Volumes (see page 863) .

With Vsphere clusters, `dkp delete` doesn't delete the virtual disks backing the PVs for DKP add ons. Therefore internal VMware cluster runs out of storage eventually. Theses PVs are only visible if VSAN is installed which gives users a Container Native Storage tab.

6.16.6.4.2 Delete the Workload Cluster

1. Make sure your vSphere credentials are up-to-date. Refresh the credentials using this command:

```
dkp update bootstrap credentials vsphere --kubeconfig $HOME/.kube/config
```

2. To delete a cluster, you would use `dkp delete cluster` and pass in the name of the cluster you are trying to delete with `--cluster-name` flag. You would use `kubectl get clusters` to get those details (`--cluster-name` and `--namespace`) of the Kubernetes cluster to delete it. NOTE: Do not use `dkp get clusters` since that gets you Kommander cluster details rather than Konvoy kubernetes cluster details.

```
kubectl get clusters
```

3. Delete the Kubernetes cluster and wait a few minutes:

   Before deleting the cluster, DKP deletes all Services of type LoadBalancer on the cluster.

To skip this step, use the flag `--delete-kubernetes-resources=false`.

```
dkp delete cluster --cluster-name=${CLUSTER_NAME} --kubeconfig $HOME/.kube/
config
```

```
INFO[2022-03-30T11:53:42-07:00] Running cluster delete command
clusterName=d2iq-e2e-cluster-1 managementClusterKubeconfig= namespace=default
 src="cluster/delete.go:95"
INFO[2022-03-30T11:53:42-07:00] Waiting for cluster to be fully deleted
src="cluster/delete.go:123"
INFO[2022-03-30T12:14:03-07:00] Deleted default/d2iq-e2e-cluster-1 cluster
src="cluster/delete.go:129"
```

6.16.6.4.3 Delete the Bootstrap Cluster

1. After the workload cluster is deleted, delete the bootstrap cluster with the following command.

```
dkp delete bootstrap --kubeconfig $HOME/.kube/config
```

```
INFO[2021-06-09T12:15:20-07:00] Deleting bootstrap cluster
src="bootstrap/bootstrap.go:182"
```

6.16.6.4.4 Next Step:

Once your cluster is built in the Konvoy component of DKP for your infrastructure/environment, you will install the Kommander (see page 1558) component of DKP to see your dashboard and continue customization.

6.16.6.4.5 Known limitations

Be aware of these limitations in the current release of DKP Konvoy.

- The DKP Konvoy version used to create the workload cluster must match the DKP Konvoy version used to delete the workload cluster.

6.17 VMware Cloud Director Infrastructure

VMware Cloud Director (VCD) is a cloud services platform that delivers virtual data center computation, network, storage, and security in a self-service model for tenants.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

6.17.1 Special Resource Requirements for VMware Cloud Director

You can control the virtual machine (VM) resource allocation and placement on a specific cluster or host by using VM sizing policies, VM placement policies, and vGPU policies.

6.17.1.1 VM Sizing Policy - CPU and Memory

For CPU and Memory, the VCD **Provider** creates the appropriate VM Sizing Policies. The VM Sizing Policy defines CPU and memory resources available to the VM.

1. When creating the cluster, reference these VM Sizing Policies by using the `--control-plane-sizing-policy` and `--worker-sizing-policy` flags.
2. See https://docs.vmware.com/en/VMware-Cloud-Director/10.4/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-73C48A9C-79AF-402C-9746-4C81CAC2B35A.html regarding parameters like *vCPU Speed* or *CPU Reservation Guarantee* that require consideration in VM Sizing Policy.

6.17.1.2 VM Placement Policy

The VCD Provider(SP) determines the VM Placement Policy[1026]. This policy defines the infrastructure where the VM should run meaning its placement of a VM on a host. When you assign a **VM Placement Policy** to a virtual machine, the placement engine adds this virtual machine to the corresponding VM group of the cluster on which it resides.

6.17.1.3 Storage

For storage, follow Create a Base OS image in vSphere vCenter (see page 1366) when creating the KIB base image. In VCD, the base image determines the minimum storage available for the VM.

1026 https://docs.vmware.com/en/VMware-Cloud-Director/10.4/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-236A070E-83E6-4648-8F2F-557248C9735D.html

For more information about VCD parameters that can affect performance, refer to VMware Documentation: https://docs.vmware.com/en/VMware-Cloud-Director/10.4/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-73C48A9C-79AF-402C-9746-4C81CAC2B35A.html

The following procedure advises on Organization setup as it relates to installing DKP and creating clusters on VCD.

- Cloud Director for Service Providers (see page 1445)
    - Cloud Director Prerequisites (see page 1446)
    - Cloud Director Configure the Organization (see page 1452)
    - Cloud Director Install DKP (see page 1461)
    - Cloud Director Management Tools (see page 1476)

More information can be found directly in the VMWare Cloud Director documentation[1027].

6.17.2 Next Step

Cloud Director for Service Providers (see page 1445)

6.17.3 Cloud Director for Service Providers

VMware Cloud Director[1028] is a platform that turns physical data center into virtual data centers and runs on one or more vCenters. After you have vSphere vCenter[1029], a brief overview of the general workflow from this section of the documentation is below:

- Use vCenter to create and configure virtual infrastructure, including:
    - Creating the **Organization**[1030](tenant) and its users
    - Creating Roles, Rights, and other related permissions necessary for the tenant users and other software components
    - Creating Base Images and VM Templates that will be uploaded to the vApp catalog[1031] of the VCD tenant **Organization**
- Using DKP to create and manage clusters and node pools
- Cloud Director Prerequisites (see page 1446)
- Cloud Director Configure the Organization (see page 1452)

1027 https://docs.vmware.com/en/VMware-Cloud-Director/index.html
1028 https://docs.vmware.com/en/VMware-Cloud-Director/10.4/rn/vmware-cloud-director-104-release-notes/index.html
1029 https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.esxi.install.doc/GUID-B2F01BF5-078A-4C7E-B505-5DFFED0B8C38.html
1030 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-2F217F99-48C1-42F3-BF06-5ABBACADE2BA.html
1031 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-D5737821-C3A4-4C73-8959-CA293C12A7DE.html

- Cloud Director Install DKP (see page 1461)
- Cloud Director Management Tools (see page 1476)

To see a visual architecture structure as it relates to Tenants, Organization(or Tenant) VDCs and Provider VDCs for VMWare Cloud Director, refer to the page linked here for their documentation under the title: New to VMware Cloud Director[1032].

6.17.3.1 Related Topics:

MSP Program (see page 105)

Version Support Policy (see page 1995) - defines a Beta feature

6.17.3.2 Helpful VMWare Resources:

- Setting Up Cloud Director Environment[1033] in GitHub
- https://www.youtube.com/@VMwareCloudProvider

6.17.3.3 Next Step

Cloud Director Prerequisites (see page 1446)

6.17.3.4 Cloud Director Prerequisites

Before continuing to install DKP on VMware Cloud Director(VCD), verify that your VMware vSphere Client environment is running vSphere Client version[1034]v6.7.x with Update 3 or later version with ESXi[1035]. You must be able to reach the vSphere API endpoint from where the Konvoy command line interface (CLI) runs. See the listed Prerequisites for all the specific details below.

6.17.3.4.1 Prerequisites Related to VMware Cloud Director

- Ensure you have meet VMware Cloud Director requirements: https://docs.vmware.com/en/VMware-Cloud-Director/10.4/rn/vmware-cloud-director-104-release-notes/index.html

1032 https://docs.vmware.com/en/VMware-Cloud-Director/index.html
1033 https://github.com/mesosphere/cluster-api-provider-cloud-director/blob/fc216d559ff875da8ee0c61cf82454b759bb16aa/docs/VCD_SETUP.md#L44
1034 https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.vsphere.vm_admin.doc/GUID-55238059-912E-411F-A0E9-A7A536972A91.html
1035 https://docs.vmware.com/en/VMware-vSphere/index.html

- Download and install VMware Cloud Director: Download VMware Cloud Director[1036]
- Considerations regarding CPU and Memory during configuration:
    - For CPU and Memory, the VCD **Provider** creates the appropriate *VM Sizing Policies*.
    - The **Provider** (or tenant user with proper permissions) references these *VM Sizing Policies* when creating the cluster, using the `--control-plane-sizing-policy` and `--worker-sizing-policy` flags.
    - See https://docs.vmware.com/en/VMware-Cloud-Director/10.4/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-73C48A9C-79AF-402C-9746-4C81CAC2B35A.html regarding parameters like *vCPU Speed* or *CPU Reservation Guarantee* that require consideration in VM Sizing Policy. For example, vCPU minimum speed is recommended to be at least 3 GHz.
- Considerations for Storage during configuration:
    - For storage, follow Create a Base OS image in vSphere (see page 1366) when creating the KIB base image. In VCD, the base image determines the minimum storage available for the VM.

> Values required to create *VM Sizing Policies* and other fields are left to the Provider's discretion. For more information about VCD parameters that can affect performance, refer to VMware Documentation: https://docs.vmware.com/en/VMware-Cloud-Director/10.4/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-73C48A9C-79AF-402C-9746-4C81CAC2B35A.html

6.17.3.4.2 Prerequisites for DKP

Before beginning a DKP installation, verify that you have:

- An x86_64-based Linux or macOS machine with a supported version of the operating system to execute the commands needed to run DKP and to create a cluster.
- Download (see page 76) the `dkp` binary for Linux or macOS with KIB image bundle. To check which version of DKP you installed for compatibility reasons, run the `dkp version -h` command (dkp version (see page 1943)).
- A Container engine/runtime installed is required to install DKP:
    - Version Docker®[1037] container engine version 18.09.2 or higher installed for Linux or MacOS - On macOS, Docker runs in a virtual machine which needs configured with at least 8 GB of memory.

1036 https://customerconnect.vmware.com/downloads/info/slug/datacenter_cloud_infrastructure/vmware_cloud_director/10_4

1037 https://docs.docker.com/get-docker/

    - Version 4.0 of Podman[1038] or higher for Linux. Host requirements found here: Host Requirements[1039]
- kubectl[1040] for interacting with the running cluster
- Valid VMware Cloud Director version 10.4.x[1041] or later account as mentioned in Cloud Director prerequisites
- You as the Service Provider (SP) have created the appropriate images in vSphere vCenter available for upload to the tenant organization's catalog:
    a. Use `konvoy-image build vsphere` to create one or more OVA's
    b. Create a vApp Template[1042] from the OVA
    c. Before the SP creates a cluster in a VCD tenant, make this vApp Template available to that tenant
- You must be able to reach the VCD API endpoint from where the DKP CLI runs

6.17.3.4.3 Machine Specifications

6.17.3.4.3.1 Control plane nodes

You must have at least three control plane nodes. Each control plane node should have at least:

- 4 cores
- 16 GiB memory
- Approximately 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.
- Disk usage must be below 85% on the root volume.

DKP on VCD defaults to deploying an `n2-standard-4` instance with an 80GiB root volume for control plane nodes, which meets the above requirements.

6.17.3.4.3.2 Worker nodes

You must have at least four worker nodes. The specific number of worker nodes required for your environment can vary depending on the cluster workload and size of the nodes. Each worker node should have at least:

- 8 cores
- 32 GiB memory
- Around 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.

1038 https://podman.io/getting-started/installation
1039 https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
1040 https://kubernetes.io/docs/tasks/tools/#kubectl
1041 https://docs.vmware.com/en/VMware-Cloud-Director/10.4.1/rn/vmware-cloud-director-1041-release-notes/index.html
1042 https://docs.vmware.com/en/VMware-Cloud-Director/10.4/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-D5737821-C3A4-4C73-8959-CA293C12A7DE.html?hWord=N4IghgNiBcIG4EEAOSAEAXApgWyRMWAziAL5A

- Disk usage must be below 85% on the root volume.

DKP on VCD defaults to deploying a `n2-standard-8` instance with an 80GiB root volume for worker nodes, which meets the above requirements.

6.17.3.4.3.3 More Information:

- Cloud Director Concepts and Terms (see page 1449)
- Port Permissions (see page 1450)

6.17.3.4.4 Next Step

Cloud Director Configure the Organization (see page 1452)

6.17.3.4.5 Cloud Director Concepts and Terms

Before attempting to create a DKP cluster in VMware Cloud Director(VCD), we recommend taking time to familiarize yourself with the following concepts and related terminology.

6.17.3.4.5.1 VMware Cloud Director Concepts

The VMware Cloud Director is based on the following concepts:

- **Provider:** Service Provider(SP) that administrates the data centers and provisions virtual infrastructure for Organizations(tenants).
- **Organizations (Tenants)**: A unit of administration for users, groups, and computing resources. Tenant users are managed at the **Organization** level.
- **System Administrators:** This role exists only in the Provider organization (SP) and can create and provision tenant **Organizations** in the Service Provider[1043] as well as the Organization[1044] portal. By default, the **System Administrator** role has all VMware Cloud Director rights.
- **Organization Administrators:** This role creates users, groups, and service catalogs. Tenant **Organization Administrator**[1045] is a predefined role that can use the VCD tenant portal[1046] to manage users in their **Organization** and assign them roles.
- **Rights**: Each right provides view or manage access to a particular object type in VCD. Also see: https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-816FBBBC-2CDA-4B1D-9B1A-C22BC31B46F2.html
- **Rights Bundle**: A collection of rights for the **Organization** as a whole.

1043 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-F8F4B534-49B2-43B2-AEEE-7BAEE8CE1844.html

1044 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-120992A9-4FCB-4900-B19C-9AACFCB3F40B.html

1045 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-1CACBB2E-FE35-4662-A08D-D2BCB174A43C.html

1046 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-120992A9-4FCB-4900-B19C-9AACFCB3F40B.html

- **Roles**: A role is a set of rights that is assignable to one or more users and groups. When you create or import a user or group, you must assign it a role. Also see: https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-816FBBBC-2CDA-4B1D-9B1A-C22BC31B46F2.html mentioned above or the section in DKP documentation: Cloud Director Roles, Rights and Rights Bundles (see page 1454)
- **Users and Groups**: Administrators can create users manually, programmatically, or integrate with a directory service like LDAP to import user accounts and user groups at scale.
- **Virtual Data Centers (VDC)**: An isolated environment provided to a cloud user, in which they can provision resources, deploy, store and operate applications.
- **Organization VMware Cloud Director Networks**: Similar to the Amazon concept of Virtual Private Cloud, a VMware Cloud Director network is available only to a specific VMware Cloud Director and available to all vApps in the **Organization**. It can be connected to external networks as needed.
- **vApp Networks**: Similar to the concept of a subnet, a vApp network is an isolated network within a VMware Cloud Director network that allows specific vApps to communicate with each other.
- **vApp**: One or more virtual machines(VMs) that come preconfigured to provide a specific type of cloud service. vApp is a virtual app that defines compute, storage and networking metadata.
- **Media Files and Catalogs**: VMware Cloud Director organizes deployable resources via media files. These are virtual machine and vApp templates (machine images) that can be used as a sort of initial boot program for a VM. The **Organization Administrator** organizes these files into catalogs, allowing users within the **Organization** to provision the resources they need.
- **Storage Profiles:** VCD concept to organize storage[1047] (ex. Gold, Platinum)
- **NSXT gateway:** A Logical Router configured in a traditional hardware switch. Gateways are used to provide connectivity to external networks and between different logical networks.
- **Tier-0 Gateway:** The Tier-0 gateway provides connectivity to external networks, using static routes or BGP. The Tier-0 gateway is primarily in charge of handling the North-South traffic between the virtualized environment and the external physical network.
- **Tier-1 Gateway:** The Tier-1 Gateway acts as a tenant router. The Tier-1 gateway is optimized for East-West traffic.
- **Edge Gateway:** A gateway that provides a VDC with connectivity and other features. It can provide NAT, firewall and other network features.
- **Provider Gateway:** A logical gateway representing Tier-0 gateway and managed by the SP.
- **Organization Edge Gateway:** An organization specific gateway in the Provider Gateway. They are created using network segments in the provider Gateway.

6.17.3.4.5.2 Next Step:

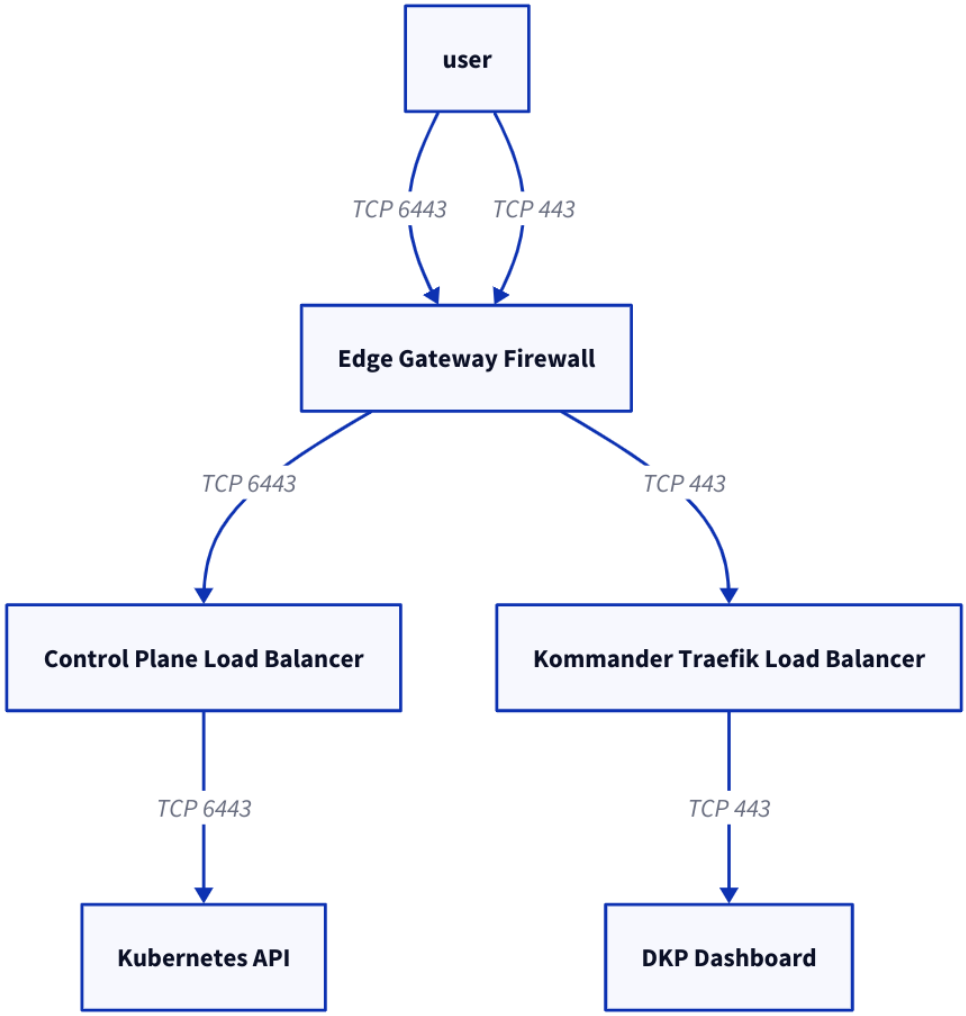
Cloud Director Configure the Organization (see page 1452)

6.17.3.4.6 Port Permissions

There are various VCD abstractions such as Edge Gateway Firewall Rules, IP sets, and Security Groups that work together. For many Kubernetes components in your DKP cluster, access is required. Allow access to

1047 https://docs.vmware.com/en/VMware-Cloud-Director/10.5/VMware-Cloud-Director-Tenant-Guide/GUID-17DDD3AC-0ABB-49EC-B3AE-28DFB2D2B80B.html?hWord=N4IghgNiBcIMoBcD2AnMBzApgAgMITAGdCQBfIA

ports 6443 and 443 from outside the cluster. There are other ports listed in the DKP ports (see page 65) page that must be allowed among the machines in the cluster.



6.17.3.5 Cloud Director Configure the Organization

The following steps document the minimum configuration needed when creating and configuring an **Organization**(tenant) for use with DKP. Refer to VMware's VCD documentation[1048] for additional tenant **Organization** configuration information.

The two main sections of the VMware Cloud Director documentation are for system administration and the tenant configuration and access:

- **Service Provider** Admin Portal - https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-F8F4B534-49B2-43B2-AEEE-7BAEE8CE1844.html
- **Organization** (tenant) Portal - https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-120992A9-4FCB-4900-B19C-9AACFCB3F40B.html

6.17.3.5.1 Prerequisites to Create and Configure a Organization

1. Add a vCenter Server[1049] to VCD - Log in to the Cloud Director portal, select the **Resources** tab across the top menu - **Infrastructure Resources** - **vCenter Server Instances** and select <ADD> to begin the process of connecting your vCenter server[1050] by following the on screen instructions.
2. Create **Organization**[1051] (Tenant) - You create a new organization[1052] from the VMware Cloud Director Admin Portal.

6.17.3.5.2 Configuring the Organization

1. Create the **Organization's** VDCs[1053]
2. Configure Edge Gateway[1054]
3. Create the Tenant Network[1055]

1048 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-1AE706A9-79F2-4A0E-8B0D-FFE9E87109CD.html

1049 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-F107933D-0E66-497E-BCB1-34A670E79D7C.html

1050 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-88470D75-4899-45DF-B01D-49C847CA4945.html

1051 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-03E72753-1BEB-45C0-BAF8-AC71A089F14E.html

1052 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-03E72753-1BEB-45C0-BAF8-AC71A089F14E.html

1053 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-09851831-142E-46B9-A278-6488784D5B6A.html

1054 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-45C0FEDF-84F2-4487-8DB8-3BC281EB25CD.html

1055 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-8F806B38-2489-4D36-82FF-B23BAFC3B294.html

4. Configure Policies[1056]

After a tenant **Organization** is created, use the menus in the **System Administrator** portal to configure the following settings:

1. Under the Data Center tab - **Virtual Data Center**: this is the location in which to define CPU size, memory and storage.
2. Under the Data Center tab - **Networking - Edges**: select **Configuration**-**External Networks** and supply the publicly accessible IP address range under Subnet column.
3. Under the Libraries tab - **Content Libraries** in left menu: specify the vApp Templates to import the VM Templates from vCenter that you want to make available to the tenant. EX: KIB templates from vCenter
   NOTE: The Service Provider(SP) can create a shared *Catalog* where items are placed to be automatically imported.
4. Under the Networking tab - **Networks**: the tenant **Organization Administrator** will configure a network. Select the name of the network to be taken to its *General* properties such as Gateway CIDR address where all VMs will receive an IP address from the private IP space.
   NOTE: The LoadBalancer (LB) will use the routable external network and is automatically created using the CAPVCD controller.
5. Under the Resources tab - **Edge Gateway** - **Services - NAT** - External IP: the Specify the IP address that will be used to allow VMs to access external networks. You should either create a SNAT rule or provide Egresses to the VM's.
6. **Edge Gateway Firewall** - Allow port access from outside the cluster for TCP 6443 to the control plane endpoint load balancers and TCP 443 to Kubernetes load balancers (e.g. to reach the DKP dashboard). Other ports must be allowed access among the machines in the cluster.
   For more DKP Ports, refer to that section under Architecture DKP Ports (see page 65) .

The tenant is required to get one public IP to create an Edge Gateway. The Service Provider(SP) will allocate the pool of IPs from which the tenant pulls. After you have associated an external network though a gateway, the tenant can ask for IPs. Otherwise, if you have chosen an IP address, you can specify it.

6.17.3.5.3 Production Policies

After the tenant **Organization** is in production, various policies will need to be defined for storage and resources. The Configure **Organization** Policy Section[1057] of VMware documentation will provide more detail.

1056 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-DBA11D01-E102-47A4-926C-BFDB681B75F2.html

1057 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-DBA11D01-E102-47A4-926C-BFDB681B75F2.html

6.17.3.5.4 Next Step

Cloud Director Roles, Rights and Rights Bundles (see page 1454)

6.17.3.5.5 Cloud Director Roles, Rights and Rights Bundles

VMware Cloud Director(VCD) has a variety of ways to assign permissions. As a Service Provider (Cloud Provider), you have numerous options when it comes to giving tenants access to VMware Cloud Director features. There are:

- Rights[1058] - provide view or manage access to a particular object type in VMware Cloud Director and belong to different categories depending on the objects to which they relate such as vApp, Catalog, or an Organization
- Roles[1059] - a collection of rights for a User and defines what an individual user has access to
- Rights Bundles[1060] - a collection of rights for the tenant **Organization** as a whole and defines what a tenant **Organization** has access to

Various **Rights** are common to multiple predefined Global Roles. These **Rights** are granted by default to all new organizations, and are available for use in other **Roles** created by the tenant **Organization Administrator**. There are some predefined **Roles** for both Provider and Tenant explained in the VMware documentation: https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-BC504F6B-3D38-4F25-AACF-ED584063754F.html.

6.17.3.5.5.1 Service Provider(SP) System Administrator

The **System Administrator**[1061] role exists only in the provider organization.

As a Service Provider(SP) you will have roles for vSphere vCenter as well as VMware Cloud Director(VCD).

- **vCenter/NXT/AVI Infrastructure System Administrator:** Manages physical infra for vCenter, NXT network fabric, AVI load balancers (D2IQ SRE / Service provider(OVH cloud))
- **System Administrator - Provider(SP) :** Manages Virtual infra in VCD that uses vCenter(s), NXT(s), AVI(s) etc. (D2IQ SRE)
- **Organization(Tenant) Administrator:** Manages Virtual infra (org, orgvdc, network, catalogs, templates, users etc) for a tenant. Users that can create k8s cluster

Through the VMware Cloud Director(VCD) Service Provider Admin Portal[1062], the SP can add **System Administrators** for Cloud Director and see the predefined list of rights included in any role. The **System Administrator** manages the virtual infrastructure in VCD that uses vCenter(s), NXT(s), AVI(s) and other components of the VCD environment.

1058 https://docs.vmware.com/en/VMware-Cloud-Director/10.4/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-9B462637-E350-43B6-989A-621F226A56D4.html

1059 https://docs.vmware.com/en/VMware-Cloud-Director/10.4/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-9B462637-E350-43B6-989A-621F226A56D4.html

1060 https://blogs.vmware.com/cloudprovider/2019/12/effective-rights-bundles.html

1061 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-438B2F8C-65B0-4895-AF40-6506E379A89D.html

1062 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-F8F4B534-49B2-43B2-AEEE-7BAEE8CE1844.html

Also, refer to Managing System Administrator and Roles[1063] documentation from the VMware documentation site.

6.17.3.5.5.2 Organization(Tenant) Administrator

As an **Organization Administrator**, from the tenant portal[1064] you can create, edit, import, and delete users. The tenant **Organization Administrator** manages the virtual infrastructure that includes the organization itself which includes the related network, catalogs, templates and such for the Tenant Organization. Important predefined role information is below:

- Tenant **Organization Administrator**[1065] is predefined and can use VCD tenant portal to create[1066] and manage users[1067] in their organization and assign them roles.
- **vApp Author**[1068] role can use catalogs and create vApps

A tenant **Organization Administrator** can access roles if allowed. They can only view the global tenant roles that a **System Administrator** has published to the organization, but cannot modify them. The **Organization Administrator** can create **custom** tenant roles with similar rights and assign them to the users within their own tenant **Organization**.

There are some predefined Global Tenant Roles as well which are explained in the VMware documentation: https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-BC504F6B-3D38-4F25-AACF-ED584063754F.html as well as the https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-AE42A8F6-868C-4FC0-B224-87CA0F3D6350.html.

6.17.3.5.5.3 Tenant Roles and Rights

Several predefined global tenant roles are described in the VMware documentation[1069] regarding which components they can access and change. User rights may be available to a role, but a rights bundle needs to be published to a tenant Organization in order for those user role permissions to work.

1. Create a Rights Bundle[1070] (a collection of rights) that includes all required Rights. See the page for the Cloud Director CAPVCD User Rights (see page 1456). (see page 1456)

1063 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-9DFB2238-23FB-4D07-B563-144AC4E9EDAF.html
1064 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-74C9E10D-9197-43B0-B469-126FFBCB5121.html
1065 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-BC504F6B-3D38-4F25-AACF-ED584063754F.html#GUID-BC504F6B-3D38-4F25-AACF-ED584063754F
1066 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-1CACBB2E-FE35-4662-A08D-D2BCB174A43C.html
1067 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-A358E190-BFC0-4187-9406-66E82C92564A.html
1068 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-BC504F6B-3D38-4F25-AACF-ED584063754F.html
1069 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-AE42A8F6-868C-4FC0-B224-87CA0F3D6350.html
1070 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-CFB0EFEE-0D4C-498D-A937-390811F11B8E.html

2. Create a Tenant Role[1071] that uses all these Rights.

The SP should publish these to tenant **Organizations** that need to create clusters. Assign the Tenant Role to a VCD user and then create an API Token. The API token will be used in the DKP CLI commands to authenticate the CLI to vCenter. They will assign the Role to a VCD User, create an API Token, and pass the Token to the DKP CLI.

6.17.3.5.5.4 Related Topics:

The CAPVCD provider uses a related component called CSE[1072]. Some of the permissions necessary to create a VCD cluster are defined using this component. Note that the term Role Based Access Control (RBAC)[1073] used in the CSE documentation refers ONLY to the VCD rights and permissions necessary to perform lifecycle management of Kubernetes clusters using VCD. It has no impact on the RBAC configuration of any clusters created using VCD

Role Based Access Control (RBAC)[1074] from GitHub - The RBAC in that page refers to the roles and rights required for the tenants to perform the life cycle management of Kubernetes clusters. It does not have anything to do with the RBAC inside the Kubernetes cluster itself.

6.17.3.5.5.5 Next Step

Cloud Director CAPVCD User Rights (see page 1456)

6.17.3.5.6 Cloud Director CAPVCD User Rights

CAPVCD requires specific user rights set in order for the tenant Organization to have a **Role** that can successfully execute DKP clusters, those **Rights** must be specified. When a DKP workload cluster is created, each machine needs to register with VCD Cloud Provider Interface(CPI)[1075] and get node references.

Refer to https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-816FBBBC-2CDA-4B1D-9B1A-C22BC31B46F2.html for terminology related to Rights and Roles. The remainder of this page describes what the CAPVCD User requires in relation to **Rights** and **Rights Bundles**.

6.17.3.5.6.1 CAPVCD User

CAPVCD uses the credentials (username/password or API token) of a VCD User to manage the cluster. The VCD Cloud Provider Interface(CPI)[1076] and CSI controllers also use the same credentials. This same User needs specific API permissions, known as **Rights**, for the CAPVCD, CPI, and CSI controllers to work correctly. For a User to be granted a **Right**, the User must be associated with a **Role** that consumes this **Right**, meaning the **Role** grants **Rights** to the User.

If the User belongs to an **Organization**, then the **Right** must *also* be published by the **Provider** to the tenant **Organization** in a Rights Bundle. The Rights Bundle grants Rights to the tenant **Organization**.

1071 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-0D991FCF-3800-461D-B123-FAE7CFF34216.html
1072 https://vmware.github.io/container-service-extension/cse3_1/INTRO.html
1073 https://vmware.github.io/container-service-extension/cse3_1/RBAC.html#additional-required-rights
1074 https://vmware.github.io/container-service-extension/cse3_1/RBAC.html#additional-required-rights
1075 https://github.com/vmware/cloud-provider-for-cloud-director/tree/main#terminology
1076 https://github.com/vmware/cloud-provider-for-cloud-director/tree/main#terminology

Create the VCD User Required by CAPVCD

CAPVCD requires all the Rights in the default `vApp Author` Global Role[1077], plus the **Rights** required by the VCD CPI, the VCD CSI, and some **Rights** required by CAPVCD itself.

1. A Provider administrator creates a **Rights Bundle** that enumerates all the below rights. We recommend the name `DKP Cluster Admin` for the **Rights Bundle**.
2. A Provider administrator creates a **Global Role** that enumerates all the below rights. We recommend the name `DKP Cluster Admin` for the **Global Role**.
3. A Provider administrator publishes the both the **Rights Bundle** and **Global Role** to every Organization that will deploy DKP clusters.
4. An Organization administrator creates a User, and associates it with the **Global Role**.

The procedure for Creating a Rights Bundle is found in VMware Documentation: https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-CFB0EFEE-0D4C-498D-A937-390811F11B8E.html and the steps for creating the Role: https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-0D991FCF-3800-461D-B123-FAE7CFF34216.html

6.17.3.5.6.2 List of Rights

CAPVCD requires the following **Rights**:

- The **Rights** cataloged by the predefined `vApp Author` Global **Role**.[1078]
- Additional **Rights** listed in the CAPVCD documentation[1079].
- Additional **Rights** listed in the VCD CPI documentation[1080].
- Additional **Rights** listed in the VCD CSI documentation[1081].

The majority of the **Rights** are from the `vApp Author` Global Role. When independent components, like CAPVCD and CPI, need the same **Right**, that **Right** appears in multiple sources.

Below are the lists of the **Rights** from the above sources as well as **Rights** required, but not documented by CAPVCD. The last list includes the **Rights** from all sources, with duplicates removed:

Required by the `vApp Author` Role

1077 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Service-Provider-Admin-Portal-Guide/GUID-BC504F6B-3D38-4F25-AACF-ED584063754F.html

1078 https://docs.vmware.com/en/VMware-Cloud-Director/10.4/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-AE42A8F6-868C-4FC0-B224-87CA0F3D6350.html#GUID-AE42A8F6-868C-4FC0-B224-87CA0F3D6350

1079 https://github.com/vmware/cloud-director-named-disk-csi-driver/blob/1.3.2/README.md#additional-rights-for-csi

1080 https://github.com/vmware/cloud-provider-for-cloud-director/blob/1.3.0/README.md#additional-rights-for-cpi

1081 https://github.com/vmware/cluster-api-provider-cloud-director/blob/b315c9c5e4b1b05600ccec20eb5116cc6173f845/docs/VCD_SETUP.md#publish-the-rights-to-the-tenant-organizations

Some Rights appear in more than one source. This list includes the Rights from all sources, with duplicates removed in the final list below:

```
Catalog: Add vApp from My Cloud
Catalog: View Private and Shared Catalogs
Organization vDC Compute Policy: View
Organization vDC Disk: View IOPS
Organization vDC Named Disk: Create
Organization vDC Named Disk: Delete
Organization vDC Named Disk: Edit Properties
Organization vDC Named Disk: View Encryption Status
Organization vDC Named Disk: View Properties
Organization vDC Network: View Properties
Organization vDC: VM-VM Affinity Edit
Organization: View
UI Plugins: View
vApp Template / Media: Copy
vApp Template / Media: Edit
vApp Template / Media: View
vApp Template: Checkout
VAPP_VM_METADATA_TO_VCENTER
vApp: Copy
vApp: Create / Reconfigure
vApp: Delete
vApp: Download
vApp: Edit Properties
vApp: Edit VM Compute Policy
vApp: Edit VM CPU
vApp: Edit VM Hard Disk
vApp: Edit VM Memory
vApp: Edit VM Network
vApp: Edit VM Properties
vApp: Manage VM Password Settings
vApp: Power Operations
vApp: Sharing
vApp: Snapshot Operations
vApp: Upload
vApp: Use Console
vApp: View ACL
vApp: View VM and VM's Disks Encryption Status
vApp: View VM metrics
vApp: VM Boot Options
```

Additional Rights Required by CAPVCD

```
API Tokens: Manage
Organization vDC Gateway: Configure Load Balancer
Organization vDC Gateway: Configure NAT
```

```
Organization vDC Gateway: View
vApp: Allow All Extra Config
```

Rights Required to List Catalogs (Not Documented by CAPVCD)

These Rights are not documented by CAPVCD, but are required. They allow CAPVCD to list the Catalogs in the Organization.

```
General: Administrator View
Access All Organization VDCs
```

Implied Rights (Not Documented by CAPVCD)

These Rights are not documented by CAPVCD, but are implied by the documented Rights.

```
Certificate Library: View
Organization vDC Gateway: View NAT
Organization vDC Gateway: View Load Balancer
```

Additional Rights Required by CPI

```
API Tokens: Manage
Organization vDC Gateway: Configure Load Balancer
Organization vDC Gateway: Configure NAT
Organization vDC Gateway: View
```

Additional Rights Required by CSI

```
API Tokens: Manage
Organization vDC Shared Named Disk: Create
Organization vDC Named Disk: Delete
Organization vDC Named Disk: Edit Properties
Organization vDC Named Disk: View Encryption Status
Organization vDC Named Disk: View Properties
```

All Sources Merged with Duplicates Removed

```
Access All Organization VDCs
API Tokens: Manage
Catalog: Add vApp from My Cloud
```

```
Catalog: View Private and Shared Catalogs
Certificate Library: View
General: Administrator View
Organization vDC Compute Policy: View
Organization vDC Disk: View IOPS
Organization vDC Gateway: Configure Load Balancer
Organization vDC Gateway: Configure NAT
Organization vDC Gateway: View
Organization vDC Gateway: View Load Balancer
Organization vDC Gateway: View NAT
Organization vDC Named Disk: Create
Organization vDC Named Disk: Delete
Organization vDC Named Disk: Edit Properties
Organization vDC Named Disk: View Encryption Status
Organization vDC Named Disk: View Properties
Organization vDC Network: View Properties
Organization vDC Shared Named Disk: Create
Organization vDC: VM-VM Affinity Edit
Organization: View
UI Plugins: View
vApp Template / Media: Copy
vApp Template / Media: Edit
vApp Template / Media: View
vApp Template: Checkout
VAPP_VM_METADATA_TO_VCENTER
vApp: Allow All Extra Config
vApp: Copy
vApp: Create / Reconfigure
vApp: Delete
vApp: Download
vApp: Edit Properties
vApp: Edit VM Compute Policy
vApp: Edit VM CPU
vApp: Edit VM Hard Disk
vApp: Edit VM Memory
vApp: Edit VM Network
vApp: Edit VM Properties
vApp: Manage VM Password Settings
vApp: Power Operations
vApp: Sharing
vApp: Snapshot Operations
vApp: Upload
vApp: Use Console
vApp: View ACL
vApp: View VM and VM's Disks Encryption Status
vApp: View VM metrics
vApp: VM Boot Options
```

6.17.3.5.6.3 Next Step

Cloud Director Install DKP (see page 1461)

6.17.3.6 Cloud Director Install DKP

Before continuing to install DKP on Cloud Director, verify that your VMware vSphere Client environment is running vSphere Client version[1082]v6.7.x with Update 3 or later version with ESXi[1083]. You must be able to reach the vSphere API endpoint from where the Konvoy command line interface (CLI) runs and have a vSphere account containing Administrator privileges. A RedHat® subscription is required with username and password for downloading DVD ISOs and valid vSphere values for the following: vCenter API server URL Datacenter name Zone name that contains ESXi hosts for your cluster's nodes.

6.17.3.6.1 If Required - Further vSphere Prerequisites

- vSphere: VMware Prerequisites (see page 446)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430985/vSphere+Prerequisites+-+Storage+Options (see page 445)
- https://d2iq.atlassian.net/wiki/spaces/DENT/pages/162430977/vSphere+Prerequisites+-+Minimum+User+Permissions (see page 442)

6.17.3.6.2 Begin Installing DKP

- Cloud Director Create Image and Template (see page 1461)
- Cloud Director Bootstrap the Cluster (see page 1462)
- Cloud Director Create a New Cluster (see page 1464)
- Cloud Director Make the New Cluster Self-Managed (see page 1468)
- Cloud Director Explore the Cluster (see page 1471)
- Cloud Director Install Kommander (see page 1473)

6.17.3.6.3 Cloud Director Create Image and Template

6.17.3.6.3.1 Create Image for Tenants

As a vSphere vCenter administrator, you will create the images for your VMWare Cloud Director (VCD) tenants.

1. The vSphere Base OS Image (see page 1366) and VM Template (see page 1379) are made in vCenter for Ubuntu 20.04. For storage, the base image determines the minimum storage available for the VM.
2. An OVA would be created from image for each VCD tenant accordingly.
3. That image is exported from vSphere and imported into the catalog of the VCD tenant **Organization** catalog of vApp Templates[1084].

1082 https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.vsphere.vm_admin.doc/GUID-55238059-912E-411F-A0E9-A7A536972A91.html
1083 https://docs.vmware.com/en/VMware-vSphere/index.html
1084 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-D5737821-C3A4-4C73-8959-CA293C12A7DE.html

Refer to the VMWare documentation for specifics on creating and deploying OVA for your tenants: Deploying OVF and OVA Templates[1085] and Working with vApp Templates[1086].

If you have not already done so, create your base image and template in vSphere and import into the VCD tenant catalog to make available for tenant use.

6.17.3.6.3.2 Next Step

Cloud Director Bootstrap the Cluster (see page 1462)

6.17.3.6.4 Cloud Director Bootstrap the Cluster

To create Kubernetes clusters, DKP uses Cluster API[1087] (CAPI) controllers. These controllers run on a Kubernetes cluster. To get started, you need a *bootstrap* cluster. Bootstrapping is the process of creating a new Kubernetes cluster from scratch and involves setting up the control plane and worker nodes. It also determines which node has the correct information for synchronization with all other nodes.

6.17.3.6.4.1 Prerequisites

Before you begin, you must:

- Complete the steps in Prerequisites (see page 1446).
- Ensure the `dkp` binary can be found in your `$PATH`.
- For tenants, you need to set their Organization name and Network name in order to create a cluster.

6.17.3.6.4.2 Bootstrap Cluster Lifecycle Services

1. Review Universal Configurations for all Infrastructure Providers (see page 1052) regarding settings, flags and other choices and then begin bootstrapping.
2. Create a bootstrap cluster:

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config
   ```

   - Configuring an HTTP/HTTPS Proxy (see page 1053) use `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful.

HTTP only flags:

To create a bootstrap cluster in a proxied environment (see page 1054) use these flags during the `dkp create bootstrap` command:

1085 https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.vsphere.vm_admin.doc/GUID-AFEDC48B-C96F-4088-9C1F-4F0A30E965DE.html
1086 https://docs.vmware.com/en/VMware-Cloud-Director/10.3/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-D5737821-C3A4-4C73-8959-CA293C12A7DE.html
1087 https://cluster-api.sigs.k8s.io/

```
--http-proxy <string> \
--https-proxy <string> \
--no-proxy <string>
```

3. DKP creates a bootstrap cluster using KIND[1088] as a library. DKP then deploys the following Cluster API[1089] providers on the cluster:
    - Core Provider[1090]
    - Cloud Director Infrastructure Provider[1091]
    - Kubeadm Bootstrap Provider[1092]
    - Kubeadm ControlPlane Provider[1093]
4. DKP waits until the controller-manager and webhook deployments of these providers are ready. List these deployments using this command:

    ```
    kubectl get --all-namespaces deployments -l=clusterctl.cluster.x-k8s.io
    ```

    You will then receive the following output:

    ```
    NAMESPACE                           NAME
    READY   UP-TO-DATE   AVAILABLE   AGE
    capa-system                         capa-controller-manager
    1/1     1            1           1h
    capg-system                         capg-controller-manager
    1/1     1            1           1h
    capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-manager
    1/1     1            1           1h
    capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-
    manager   1/1     1            1           1h
    capi-system                         capi-controller-manager
    1/1     1            1           1h
    cappp-system                        cappp-controller-manager
    1/1     1            1           1h
    capv-system                         capv-controller-manager
    1/1     1            1           1h
    capz-system                         capz-controller-manager
    1/1     1            1           1h
    capvcd-system                       capvcd-controller-manager
    1/1     1            1           1h
    ```

1088 https://github.com/kubernetes-sigs/kind
1089 https://cluster-api.sigs.k8s.io/
1090 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/
1091 https://github.com/vmware/cluster-api-provider-cloud-director
1092 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/bootstrap/kubeadm
1093 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/controlplane/kubeadm

```
cert-manager                              cert-manager
1/1     1            1           1h
cert-manager                              cert-manager-cainjector
1/1     1            1           1h
cert-manager                              cert-manager-webhook
1/1     1            1           1h
```

6.17.3.6.4.3 Next Step

Cloud Director Create a New Cluster (see page 1464)

6.17.3.6.5 Cloud Director Create a New Cluster

Before beginning to create your new cluster, confirm you can reach any of the IPs in the range allocated to the load balancer pool so the CAPVCD drivers can reach it during cluster creation. The bootstrap cluster will connect to the CAPVCD controller and connect to load balancer(LB) in VCD using `sshuttle`. The bootstrap IP address will connect to the load balancer IP to deploy the CNI.

6.17.3.6.5.1 Flags Specific to VMware Cloud Director Cluster Creation

When creating a VCD cluster, CPU and Memory flags are needed:

- For CPU and Memory, the VCD **Provider** creates the appropriate VM Sizing Policies. Then the **Provider** references these VM Sizing Policies when creating the cluster, using the flags:
  - `--control-plane-sizing-policy`
  - `--worker-sizing-policy`

> If the Service Provider(SP) has given a tenant user the permissions (see page 1454) to create clusters inside their own **Organization**, then that tenant user will need to reference those flags are well.

6.17.3.6.5.2 Name your Cluster

1. Give your cluster a unique name suitable for your environment. The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[1094] for more naming information.

   In Cloud Director it is critical that the name is unique, as no two clusters in the same Cloud Director account can have the same name.

1094 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

2. Set the environment variable:

```
export CLUSTER_NAME=<vcd-example>
```

To increase Docker Hub's rate limit[1095] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username=<username> --registry-mirror-password=<password>` on the `dkp create cluster` command.

6.17.3.6.5.3 Create a New Cloud Director Cluster

After you have met all the prerequisites, begin creating a cluster:

Cloud Director needs the Base OS image (see page 1652) and VM Template (see page 1654) from vSphere to be converted to a vApp Template and listed in the tenant **Organization**'s catalog.

1. Create a vApp template[1096] using the instructions from the previous section of documentation: Cloud Director Create Image and Template (see page 1461)
2. Set Refresh API token using the instructions from VMware:

```
export VCD_REFRESH_TOKEN=<REFRESH TOKEN>
```

You must provide the refresh tokens (<REFRESH TOKEN>) when provisioning a cluster. See the VMware Documentation for details: https://docs.vmware.com/en/VMware-Cloud-Director/10.4/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-A1B3B2FA-7B2C-4EE1-9D1B-188BE703EEDE.html#procedure-2

3. Create a Kubernetes cluster:
   Reference the VM Sizing Policies during cluster creation using the flags:

1095 https://docs.docker.com/docker-hub/download-rate-limit/

1096 https://docs.vmware.com/en/VMware-Cloud-Director/10.4/VMware-Cloud-Director-Tenant-Portal-Guide/GUID-D5737821-C3A4-4C73-8959-CA293C12A7DE.html?hWord=N4IghgNiBcIG4EEAOSAEAXApgWyRMWAziAL5A

- `--control-plane-sizing-policy`
- `--worker-sizing-policy`

```
dkp create cluster vcd --cluster-name ${CLUSTER_NAME} \
--site "" \
--organization "" \
--data-center "" \
--catalog "" \
--vapp-template "" \
--network "" \
--ssh-public-key-file "" \
--dry-run \
--output=yaml \
> ${CLUSTER_NAME}.yaml
```

- (Optional) If your environment uses HTTP/HTTPS proxies (see page 1053), you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful.

  ```
  --http-proxy <<http proxy list>>
  --https-proxy <<https proxy list>>
  --no-proxy <<no proxy list>>
  ```

- (Optional) Alternatively, you can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag. This will create multiple files in the specified directory which must already exist:

  ```
  --output-directory=<existing-directory>
  ```

  Description of parameters:

- `--site`: the cloud director's endpoint with the format https://VCD_HOST.
- `--organization`: the tenant Organization name
- `--data-center`: the Virtual Data Center(VDC) name
- `--catalog`: the folder name where vApp templates are located
- `--network`: your private network

  See dkp create cluster vcd (see page 1853) for Cloud Director reference to see the full list of cluster creation options and their descriptions.

4. Create the cluster from the objects. A warning will appear in the console if the resource already exists and will require you to remove the resource or update your YAML.

```
kubectl create -f ${CLUSTER_NAME}.yaml
```

If you used the `--output-directory` flag in your `dkp create .. --dry-run` step above, create the cluster from the objects by specifying the directory in the command:

```
kubectl create -f <existing-directory>
```

5. Wait for the cluster control-plane to be ready:

```
kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
timeout=20m
```

6. Run the DKP describe command to monitor the current status of the cluster:

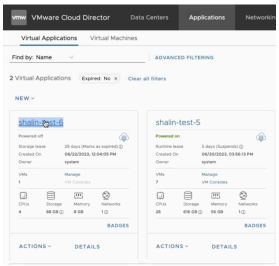
```
dkp describe cluster -c ${CLUSTER_NAME}
```

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/vcd-example                                                 True
52s
├─ClusterInfrastructure - vcdCluster/vcd-example
├─ControlPlane - KubeadmControlPlane/vcd-example-control-plane      True
52s
| ├─Machine/vcd-example-control-plane-6fbzn                         True
2m32s
| | └─MachineInfrastructure - vcdMachine/vcd-example-control-plane-62g6s
| ├─Machine/vcd-example-control-plane-jf6s2                         True
7m36s
| | └─MachineInfrastructure - vcdMachine/vcd-example-control-plane-bsr2z
| └─Machine/vcd-example-control-plane-mnbfs                         True
54s
|   └─MachineInfrastructure - vcdMachine/vcd-example-control-plane-s8xsx
└─Workers
  └─MachineDeployment/vcd-example-md-0                              True
78s
    ├─Machine/vcd-example-md-0-68b86fddb8-8glsw                     True
2m49s
    | └─MachineInfrastructure - vcdMachine/vcd-example-md-0-zls8d
```

```
    ├─Machine/vcd-example-md-0-68b86fddb8-bvbm7                    True
2m48s
    │ └─MachineInfrastructure - vcdMachine/vcd-example-md-0-5zcvc
    ├─Machine/vcd-example-md-0-68b86fddb8-k9499                    True
2m49s
    │ └─MachineInfrastructure - vcdMachine/vcd-example-md-0-k8h5p
    └─Machine/vcd-example-md-0-68b86fddb8-l6vfb                    True
2m49s
      └─MachineInfrastructure - vcdMachine/vcd-example-md-0-9h5vn
```

⚠ **NOTE**: In the Cloud Director interface, you can find this new cluster under the **Applications** tab - **Virtual Applications** (vApp). One vApp per cluster will be created into CAPVCD with virtual machine information as well as worker node and control plane node information.

EXAMPLE:



6.17.3.6.5.4 Next Step

Cloud Director Make the New Cluster Self-Managed (see page 1468)

6.17.3.6.6 Cloud Director Make the New Cluster Self-Managed

DKP deploys all cluster lifecycle services to a bootstrap cluster, which then deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, which makes the workload cluster self-managed. This section describes how to make a workload cluster self-managed.

Before starting, ensure you create a workload cluster as described in Create a New VCD Cluster (see page 575) and that you as the Managed Service Provider(MSP) have a Bastion Host (see page 1068) configured so that the CAPVCD drivers can reach it.

This page contains instructions on how to make your cluster self-managed. This is necessary if there is only one cluster in your environment, or if this cluster should become the Management cluster in a multi-cluster environment.

> If you already have a self-managed or Management cluster in your environment, skip this page.

6.17.3.6.6.1 Make the New Kubernetes Cluster Manage Itself

Follow these steps to create a Management Cluster in Enterprise or multiple stand alone Essential clusters by making the cluster manage itself:

1. Deploy cluster lifecycle services on the workload cluster:

```
dkp create capi-components --kubeconfig ${CLUSTER_NAME}.conf
```

```
✓ Initializing new CAPI components
```

2. Move the Cluster API objects from the bootstrap to the workload cluster:

   The cluster lifecycle services on the workload cluster are ready, but the workload cluster configuration is on the bootstrap cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the bootstrap to the workload cluster. This process is also called a Pivot[1097].

```
dkp move capi-resources --to-kubeconfig ${CLUSTER_NAME}.conf
```

```
✓ Moving cluster resources
You can now view resources in the moved cluster by using the --kubeconfig flag
with kubectl. For example: kubectl --kubeconfig=vcd-example.conf get nodes
```

   NOTE: To ensure only one set of cluster lifecycle services manages the workload cluster, DKP first pauses reconciliation of the objects on the bootstrap cluster, then creates the objects on the workload cluster. As DKP copies the objects, the cluster lifecycle services on the workload cluster reconcile the objects. The workload cluster becomes self-managed after DKP creates all the objects. If it fails, the `move` command can be safely retried.

3. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf wait --for=condition=ControlPlaneRead
y "clusters/${CLUSTER_NAME}" --timeout=20m
```

1097 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
cluster.cluster.x-k8s.io/vcd-example condition met
```

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

   NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use DKP with the workload cluster kubeconfig.

```
dkp describe cluster --kubeconfig ${CLUSTER_NAME}.conf -c ${CLUSTER_NAME}
```

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/vcd-example                                                 True
14s
├─ClusterInfrastructure - vcdCluster/vcd-example
├─ControlPlane - KubeadmControlPlane/vcd-example-control-plane      True
14s
│ ├─Machine/vcd-example-control-plane-6fbzn                         True
17s
│ │ └─MachineInfrastructure - vcdMachine/vcd-example-control-plane-62g6s
│ ├─Machine/vcd-example-control-plane-jf6s2                         True
17s
│ │ └─MachineInfrastructure - vcdMachine/vcd-example-control-plane-bsr2z
│ └─Machine/vcd-example-control-plane-mnbfs                         True
17s
│   └─MachineInfrastructure - vcdMachine/vcd-example-control-plane-s8xsx
└─Workers
  └─MachineDeployment/vcd-example-md-0                              True
17s
    ├─Machine/vcd-example-md-0-68b86fddb8-8glsw                     True
17s
    │ └─MachineInfrastructure - vcdMachine/vcd-example-md-0-zls8d
    ├─Machine/vcd-example-md-0-68b86fddb8-bvbm7                     True
17s
    │ └─MachineInfrastructure - vcdMachine/vcd-example-md-0-5zcvc
    ├─Machine/vcd-example-md-0-68b86fddb8-k9499                     True
17s
    │ └─MachineInfrastructure - vcdMachine/vcd-example-md-0-k8h5p
    └─Machine/vcd-example-md-0-68b86fddb8-l6vfb                     True
17s
      └─MachineInfrastructure - vcdMachine/vcd-example-md-0-9h5vn
```

5. Remove the bootstrap cluster, as the workload cluster is now self-managed:

```
dkp delete bootstrap --kubeconfig $HOME/.kube/config
✓ Deleting bootstrap cluster
```

6.17.3.6.6.2 Known Limitations

DKP only supports moving all namespaces in the cluster; DKP does not support migration of individual namespaces.

6.17.3.6.6.3 Next Step

Cloud Director Explore the Cluster (see page 1471)

6.17.3.6.7 Cloud Director Explore the Cluster

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster. Before you start, make sure you have created a workload cluster, as described in Create a New VCD Cluster (see page 1464).

6.17.3.6.7.1 Explore the new Kubernetes cluster

1. Get a kubeconfig file for the workload cluster:

   When the workload cluster is created, the cluster lifecycle services generate a kubeconfig file for the workload cluster, and write it to a *Secret*. The kubeconfig file is scoped to the cluster administrator.

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

2. Verify the API server is up :

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
   ```

   NOTE: It may take a few minutes for the Status to move to Ready while the Pod network is deployed. The Nodes' Status should change to Ready soon after the calico-node DaemonSet Pods are Ready. You should receive the following output:

   ```
   NAME                              STATUS   ROLES                  AGE
   VERSION
   vcd-example-control-plane-9z77w   Ready    control-plane,master   4m44s
   v1.28.7
   vcd-example-control-plane-rtj9h   Ready    control-plane,master   104s
   v1.28.7
   vcd-example-control-plane-zbf9w   Ready    control-plane,master   3m23s
   v1.28.7
   vcd-example-md-0-88c46            Ready    <none>                 3m28s   v1.28
   .7
   vcd-example-md-0-fp8s7            Ready    <none>                 3m28s   v1.28
   .7
   ```

```
vcd-example-md-0-qvnx7            Ready    <none>                 3m28s   v1.28
.7
vcd-example-md-0-wjdrg            Ready    <none>                 3m27s   v1.28
.7
```

3. List the Pods with the command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get pods -A
```

4. The output has been shortened, but will resemble the following:

```
NAMESPACE                           NAME
READY   STATUS    RESTARTS       AGE
calico-system                       calico-kube-controllers-577c696df9-v2nzv
1/1     Running   0              5m23s
calico-system                       calico-node-4x5rk
1/1     Running   0              4m22s
calico-system                       calico-node-cxsgc
1/1     Running   0              4m23s
calico-system                       calico-node-dvlnm
1/1     Running   0              4m23s
calico-system                       calico-node-h6nlt
1/1     Running   0              4m23s
calico-system                       calico-node-jmkwq
1/1     Running   0              5m23s
calico-system                       calico-node-tnf54
1/1     Running   0              4m18s
calico-system                       calico-node-v6bwq
1/1     Running   0              2m39s
calico-system                       calico-typha-6d8c94bfdf-dkfvq
1/1     Running   0              5m23s
calico-system                       calico-typha-6d8c94bfdf-fdfn2
1/1     Running   0              3m43s
calico-system                       calico-typha-6d8c94bfdf-kjgzj
1/1     Running   0              3m43s
capa-system                         capa-controller-manager-6468bc488-w7nj9
1/1     Running   0              67s
capg-system                         capg-controller-manager-5fb47f869b-6jgms
1/1     Running   0              53s
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-
manager-65ffc94457-7cjdn      1/1     Running   0              74s
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-
manager-bc7b688d4-vv8wg   1/1     Running   0              72s
capi-system                         capi-controller-manager-dbfc7b49-dzvw8
1/1     Running   0              77s
cappp-system                        cappp-controller-manager-8444d67568-rmms2
1/1     Running   0              59s
capv-system                         capv-controller-manager-58b8ccf868-rbscn
1/1     Running   0              56s
```

```
capz-system                         capz-controller-manager-6467f986d8-dnvj4
1/1     Running   0             62s
capvcd-system                       capvcd-controller-manager-dbfc7b49-dzvw8
1/1     Running   0             77s
cert-manager                        cert-manager-6888d6b69b-7b7m9
1/1     Running   0             91s
kube-system                         cluster-autoscaler-7f695dc48f-v5kvh
1/1     Running   0             5m40s
kube-system                         coredns-64897985d-hbkqd
1/1     Running   0             5m38s
kube-system                         coredns-64897985d-m8g5j
1/1     Running   0             5m38s
kube-system                         etcd-vcd-example-control-plane-9z77w
1/1     Running   0             5m32s
kube-system                         etcd-vcd-example-control-plane-rtj9h
1/1     Running   0             2m37s
kube-system                         etcd-vcd-example-control-plane-zbf9w
1/1     Running   0             4m17s
kube-system                         kube-apiserver-vcd-example-control-
plane-9z77w                 1/1     Running   0             5m32s
kube-system                         kube-controller-manager-vcd-example-
control-plane-9z77w         1/1     Running   0             5m33s
kube-system                         kube-proxy-gdkn5
1/1     Running   0             4m23s
kube-system                         kube-scheduler-vcd-example-control-plane-
zbf9w                   1/1     Running   0             4m17s
node-feature-discovery              node-feature-discovery-master-7d5985467-
lh7dc                   1/1     Running   0             5m40s
tigera-operator                     tigera-operator-5f9bdc5c59-j9tnr
1/1     Running   0             5m38s
```

6.17.3.6.7.2 Next Step

Cloud Director Install Kommander (see page 1473)

6.17.3.6.8 Cloud Director Install Kommander

The Kommander component of DKP can be configured differently depending on your environment type and other desired customizations. This installation allows access to the DKP Dashboard for cluster management and insights.

6.17.3.6.8.1 Installing Kommander by Environment

This section contains instructions for installing the Kommander component on *air-gapped*, *non-air-gapped*, or *pre-provisioned* environments. It also contains instructions on how to enable *DKP catalog apps* or *DKP Insights*, if desired.

6.17.3.6.8.2 Kommander Customizations

This section contains instructions to enable a custom configuration of a Kommander component such as custom domains or certificates, an HTTP Proxy, an external load balancer, etc.

6.17.3.6.8.3 Log in to the UI with Kommander:

After the Kommander component is installed according to the environment you selected, log in to the UI.

6.17.3.6.8.4 By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|
base64decode}}{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://
{{with index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/
dkp/kommander/dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider. Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password:
kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers:

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

6.17.3.6.8.5 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

6.17.3.6.8.6 Next Step

Day 2 - Cluster Operations Management (see page 590)

6.17.3.7 Cloud Director Management Tools

After cluster creation and configuration, you can revisit clusters to update and change variables.

- Cloud Director Cluster Autoscaler (see page 1476)
- Cloud Director Manage Node Pools (see page 1478)
- Cloud Director Delete a Cluster (see page 1483)

6.17.3.7.1 Cloud Director Cluster Autoscaler

Cluster Autoscaler[1098] provides the ability to automatically scale-up or scale-down the number of worker nodes in a cluster, based on the number of pending pods to be scheduled. Running the Cluster Autoscaler is optional.

Unlike Horizontal-Pod Autoscaler[1099], Cluster Autoscaler does not depend on any Metrics server and does not need Prometheus or any other metrics source.

The Cluster Autoscaler looks at the following annotations on a MachineDeployment to determine its scale-up and scale-down ranges:

```
cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size
cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size
```

The full list of command line arguments to the Cluster Autoscaler controller is on the Kubernetes public GitHub repository[1100].

For more information about how Cluster Autoscaler works, see these documents:

- What is Cluster Autoscaler[1101]
- How does scale-up work[1102]
- How does scale-down work[1103]
- CAPI Provider for Cluster Autoscaler[1104]

6.17.3.7.1.1 Cluster Autoscaler Prerequisites

Before you begin, you must have:

1098 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi
1099 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-fast-is-hpa-when-combined-with-ca
1100 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-are-the-parameters-to-ca
1101 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-is-cluster-autoscaler
1102 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-up-work
1103 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-down-work
1104 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi

- A Bootstrap Cluster Lifecycle (see page 1462).
- A New Kubernetes Cluster (see page 1464).
- A Self-Managed Cluster (see page 1468).

6.17.3.7.1.2 Run Cluster Autoscaler

The Cluster Autoscaler controller runs on the workload cluster. Upon creation of the workload cluster, this controller does not have all the objects required to function correctly until after a `dkp move` is issued from the bootstrap cluster.

Run the following steps to enable Cluster Autoscaler:

1. Ensure the Cluster Autoscaler controller is up and running (no restarts and no errors in the logs)

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf logs deployments/cluster-autoscaler
   cluster-autoscaler -n kube-system -f
   ```

2. Enable Cluster Autoscaler by setting the min & max ranges

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment $
   {NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size=2
   kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment $
   {NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size=6
   ```

3. The Cluster Autoscaler logs will show that the worker nodes are associated with node-groups and that pending pods are being watched.
4. To demonstrate that it is working properly, create a large deployment which will trigger pending pods (For this example we used Cloud Director n2-standard-8 worker nodes. If you have larger worker-nodes, you should scale up the number of replicas accordingly).

   ```
   cat <<EOF | kubectl --kubeconfig=${CLUSTER_NAME}.conf apply -f -
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: busybox-deployment
     labels:
       app: busybox
   spec:
     replicas: 600
     selector:
       matchLabels:
         app: busybox
     template:
       metadata:
         labels:
           app: busybox
   ```

```
    spec:
      containers:
      - name: busybox
        image: busybox:latest
        command:
          - sleep
          - "3600"
        imagePullPolicy: IfNotPresent
      restartPolicy: Always
EOF
```

5. Cluster Autoscaler will scale up the number of Worker Nodes until there are no pending pods.
6. Scale down the number of replicas for `busybox-deployment`.

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf scale --replicas=30 deployment/
busybox-deployment
```

7. Cluster Autoscaler starts to scale down the number of Worker Nodes after the default timeout of 10 minutes.

6.17.3.7.2 Cloud Director Manage Node Pools

Node pools are part of a cluster and managed as a group, and you can use a node pool to manage a group of machines using the same common properties. When Konvoy creates a new default cluster, there is one node pool for the worker nodes and all nodes in that new node pool have the same configuration. You can create additional node pools for more specialized hardware or configuration. For example, if you want to tune your memory usage on a cluster where you need maximum memory for some machines and minimal memory on other machines, you would create a new node pool with those specific resource needs.

Konvoy implements node pools using Cluster API MachineDeployments[1105].

- Cloud Director Create Node Pools (see page 1478)
- Cloud Director List Node Pools (see page 1480)
- Cloud Director Scale Node Pools (see page 1480)
- Cloud Director Delete Node Pools (see page 1482)

6.17.3.7.2.1 Cloud Director Create Node Pools

Creating a node pool is useful when you need to run workloads that require machines with specific resources, such as additional memory, or specialized network or storage hardware.

Prerequisites

Before you begin, make sure you have created a New VCD Cluster. (see page 1464)

1105 https://cluster-api.sigs.k8s.io/developer/architecture/controllers/machine-deployment.html

Prepare the Environment

Follow these steps:

1. Set the environment variable to the name you assigned this cluster.

   ```
   export CLUSTER_NAME=vcd-example
   ```

2. If your workload cluster is self-managed, as described in Make the New Cluster Self-Managed (see page 1468), configure `kubectl` to use the kubeconfig for the cluster.

   ```
   export KUBECONFIG=${CLUSTER_NAME}.conf
   ```

3. Define your node pool name.

   ```
   export NODEPOOL_NAME=example
   ```

Create a VCD Node Pool

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools is to ensure your cluster has nodes deployed in multiple Availability Zones.

Create a new AWS node pool with 3 replicas using this command:

Set the `--zone` flag to a zone[1106] in the same same region as your cluster.

```
dkp create nodepool vcd ${NODEPOOL_NAME} \
    --cluster-name=${CLUSTER_NAME} \
    --image $IMAGE_NAME \
    --zone us-west1-b \
    --replicas=3
```

```
machinedeployment.cluster.x-k8s.io/example created
⋅⋅ Creating default/example nodepool resources
vcdmachinetemplate.infrastructure.cluster.x-k8s.io/example created
kubeadmconfigtemplate.bootstrap.cluster.x-k8s.io/example created
 ✓ Creating default/example nodepool resources
```

This example uses default values for brevity. Use flags to define custom instance types, and other properties.

1106 https://cloud.google.com/compute/docs/regions-zones

Advanced users can use a combination of the `--dry-run` and `--output=yaml` or or `--output-directory=<your-target-directory>/` flags to get a complete set of node pool objects to modify locally or store in version control.

Next Topic:

Cloud Director List Node Pools (see page 1480)

6.17.3.7.2.2 Cloud Director List Node Pools

Use this command to list the node pools of a given cluster. This returns specific properties of each node pool so that you can see the name of the MachineDeployments.

To list all node pools for a managed cluster, run:

```
dkp get nodepool --cluster-name=${CLUSTER_NAME}
```

The expected output is similar to the following example, indicating the desired size of the node pool, the number of replicas ready in the node pool, and the Kubernetes version those nodes are running:

```
NODEPOOL                    DESIRED             READY               KUBERNETES
VERSION
example                     3                   3                   v1.27.6
vcd-example-2-md-0          4                   4                   v1.27.6
```

Next Topic:

Cloud Director Scale Node Pools (see page 1480)

6.17.3.7.2.3 Cloud Director Scale Node Pools

While you can run Cluster Autoscaler[1107], you can also manually scale your node pools up or down when you need more finite control over your environment. For example, if you require 10 machines to run a process, you can manually set the scaling to run those 10 machines only. However, if also using the Cluster Autoscaler, you must stay within your minimum and maximum bounds.

Scaling Up Node Pools

To scale up a node pool in a cluster, run:

```
dkp scale nodepool ${NODEPOOL_NAME} --replicas=5 --cluster-name=${CLUSTER_NAME}
```

Your output should be similar to this example, indicating the scaling is in progress:

1107 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi

```
✓ Scaling node pool example to 5 replicas
```

After a few minutes, you can list the node pools to:

```
dkp get nodepool --cluster-name=${CLUSTER_NAME}
```

Your output should be similar to this example, with the number of DESIRED and READY replicas increased to 5:

```
NODEPOOL                        DESIRED             READY
KUBERNETES VERSION
example                         5                   5                   v1.28.7
vcd-example-md-0                4                   4                   v1.28.7
```

Scaling Down Node Pools

To scale down a node pool, run:

```
dkp scale nodepool ${NODEPOOL_NAME} --replicas=4 --cluster-name=${CLUSTER_NAME}
```

```
✓ Scaling node pool example to 4 replicas
```

After a few minutes, you can list the node pools using this command:

```
dkp get nodepool --cluster-name=${CLUSTER_NAME}
```

Your output should be similar to this example, with the number of DESIRED and READY replicas decreased to 4:

```
NODEPOOL                        DESIRED             READY
KUBERNETES VERSION
example                         4                   4                   v1.28.7
vcd-example-md-0                4                   4                   v1.28.7
```

In a default cluster, the nodes to delete are selected at random. This behavior is controlled by CAPI's delete policy[1108]. However, when using the DKP CLI to scale down a node pool, it is also possible to specify the Kubernetes Nodes you want to delete.

To do this, set the flag `--nodes-to-delete` with a list of nodes as below. This adds an annotation `cluster.x-k8s.io/delete-machine=yes` to the matching Machine object that contains `status.NodeRef` with the node names from `--nodes-to-delete`.

1108 https://github.com/kubernetes-sigs/cluster-api/blob/v0.4.0/api/v1alpha4/machineset_types.go#L85-L105

```
dkp scale nodepool ${NODEPOOL_NAME} --replicas=3 --cluster-name=${CLUSTER_NAME}
```

```
✓ Scaling node pool example to 3 replicas
```

Scaling Node Pools When Using Cluster Autoscaler

If you configured the cluster autoscaler[1109] for the `demo-cluster-md-0` node pool, the value of `--replicas` must be within the minimum and maximum bounds.

For example, assuming you have these annotations:

```
kubectl annotate machinedeployment ${NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-
autoscaler-node-group-min-size=2
kubectl annotate machinedeployment ${NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-
autoscaler-node-group-max-size=6
```

Try to scale the node pool to 7 replicas with the command:

```
dkp scale nodepool ${NODEPOOL_NAME} --replicas=7 --cluster-name=${CLUSTER_NAME}
```

Which results in an error similar to:

```
 ✗ Scaling node pool example to 7 replicas
failed to scale nodepool: scaling MachineDeployment is forbidden: desired replicas 7
 is greater than the configured max size annotation cluster.x-k8s.io/cluster-api-
autoscaler-node-group-max-size: 6
```

Similarly, scaling down to a number of replicas less than the configured `min-size` also returns an error.

Next Topic:

Cloud Director Delete Node Pools (see page 1482)

6.17.3.7.2.4 Cloud Director Delete Node Pools

Deleting a node pool deletes the Kubernetes nodes and the underlying infrastructure. All nodes will be drained prior to deletion and the pods running on those nodes will be rescheduled.

To delete a node pool from a managed cluster, run:

1109 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi

```
dkp delete nodepool ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME}
```

Here, `example` is the node pool to be deleted.

The expected output will be similar to the following example, indicating the node pool is being deleted:

```
✓ Deleting default/example nodepool resources
```

Deleting an invalid node pool results in output similar to this example:

```
dkp delete nodepool ${CLUSTER_NAME}-md-invalid --cluster-name=${CLUSTER_NAME}
```

```
MachineDeployments or MachinePools.infrastructure.cluster.x-k8s.io "no
MachineDeployments or MachinePools found for cluster vcd-example" not found
```

6.17.3.7.3 Cloud Director Delete a Cluster

6.17.3.7.3.1 Prepare to Delete a Self-managed Workload Cluster

> A **self-managed** cluster cannot delete itself. If your workload cluster is self-managed, you must create a bootstrap cluster and move the cluster lifecycle services to the bootstrap cluster before deleting the workload cluster.

If you did not make your workload cluster self-managed, as described in Make the New Cloud Director Cluster Self-Managed (see page 1468), proceed to the Delete the workload cluster (see page 0) section below.

1. Create a bootstrap cluster:

   The bootstrap cluster will host the Cluster API controllers that reconcile the cluster objects marked for deletion:

   NOTE: To avoid using the wrong `kubeconfig`, the following steps use explicit `kubeconfig` paths and contexts.

   ```
   dkp create bootstrap --vcd-bootstrap-credentials=true --kubeconfig $HOME/.kube/
   config
   ```

   The output resembles:

```
✓ Creating a bootstrap cluster
✓ Initializing new CAPI components
```

2. Move the Cluster API objects from the workload to the bootstrap cluster: The cluster lifecycle services on the bootstrap cluster are ready, but the workload cluster configuration is on the workload cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the workload to the bootstrap cluster. This process is also called a Pivot[1110].

```
dkp move capi-resources \
    --from-kubeconfig ${CLUSTER_NAME}.conf \
    --to-kubeconfig $HOME/.kube/config
```

3. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

```
dkp describe cluster --kubeconfig $HOME/.kube/config -c ${CLUSTER_NAME}
```

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/vcd-example                                                 True
34s
├─ClusterInfrastructure - vcdCluster/vcd-example
├─ControlPlane - KubeadmControlPlane/vcd-example-control-plane      True
34s
| ├─Machine/vcd-example-control-plane-6fbzn                         True
37s
| | └─MachineInfrastructure - vcdMachine/vcd-example-control-plane-62g6s
| ├─Machine/vcd-example-control-plane-jf6s2                         True
37s
| | └─MachineInfrastructure - vcdMachine/vcd-example-control-plane-bsr2z
| └─Machine/vcd-example-control-plane-mnbfs                         True
37s
|   └─MachineInfrastructure - vcdMachine/vcd-example-control-plane-s8xsx
└─Workers
  └─MachineDeployment/vcd-example-md-0                              True
37s
    ├─Machine/vcd-example-md-0-68b86fddb8-8glsw                     True
37s
    | └─MachineInfrastructure - vcdMachine/vcd-example-md-0-zls8d
    ├─Machine/vcd-example-md-0-68b86fddb8-bvbm7                     True
37s
    | └─MachineInfrastructure - vcdMachine/vcd-example-md-0-5zcvc
```

1110 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
    ├─Machine/vcd-example-md-0-68b86fddb8-k9499                          True
37s
    │ └─MachineInfrastructure - vcdMachine/vcd-example-md-0-k8h5p
    └─Machine/vcd-example-md-0-68b86fddb8-l6vfb                          True
37s
      └─MachineInfrastructure - vcdMachine/vcd-example-md-0-9h5vn
```

NOTE: After moving the cluster lifecycle services to the workload cluster, remember to use `dkp` with the workload cluster kubeconfig.

6.17.3.7.3.2 Delete the Workload Cluster

1. Evict all workloads that use Persistent Volumes.
   We recommend using `dkp` with the bootstrap cluster to delete all worker node pools. This evicts all workloads that use Persistent Volumes. See information on Cloud Director Delete Node Pools (see page 1482).

> Persistent Volumes (PVs) are not deleted automatically by design in order to preserve your data. However, they take up storage space if not deleted. You must delete PVs manually. Information for backup of a cluster and PVs is on the page in documentation called Back up your Cluster's Applications and Persistent Volumes (see page 863) .

2. Use `dkp` with the bootstrap cluster to delete the workload cluster. Delete the Kubernetes cluster and wait a few minutes:

   Before deleting the cluster, DKP deletes all Services of type LoadBalancer on the cluster. To skip this step, use the flag `--delete-kubernetes-resources=false`.

   ```
   dkp delete cluster --kubeconfig $HOME/.kube/config --cluster-name $
   {CLUSTER_NAME}
   ```

   ```
   ✓ Deleting Services with type LoadBalancer for Cluster default/vcd-example
   ✓ Deleting ClusterResourceSets for Cluster default/vcd-example
   ✓ Deleting cluster resources
   ✓ Waiting for cluster to be fully deleted
   Deleted default/vcd-example cluster
   ```

   After the workload cluster is deleted, delete the bootstrap cluster.

6.17.3.7.3.3 Delete the Bootstrap Cluster

1. Delete the bootstrap cluster using `dkp delete`:

```
dkp delete bootstrap --kubeconfig $HOME/.kube/config
```

```
✓ Deleting bootstrap cluster
```

6.17.3.7.3.4 Delete a Managed Cluster

In the CLI, using the `kubeconfig` for the **Managed** cluster that you want to delete. In order to properly delete a **Managed** cluster, you must first delete the PVCs using these steps:

1. Export your workspace namespace:

   ```
   export WORKSPACE_NAMESPACE=<your-workspace-namespace>
   ```

2. Get all of the PVCs that you need to delete from the Applications deployed by Kommander:

   ```
   kubectl get pvc --namespace ${WORKSPACE_NAMESPACE}
   ```

3. Delete those PVCs and the Managed cluster deletion process will continue:

   ```
   kubectl delete pvc --namespace ${WORKSPACE_NAMESPACE} <pvc-name-1> <pvc-name-2>
   ```

6.17.3.7.3.5 Known Limitations

The DKP version used to create the workload cluster must match the DKP version used to delete the workload cluster.

6.17.3.7.3.6 Next Step:

Day 2 - Cluster Operations Management (see page 590)

6.18 GCP Infrastructure

The following procedure describes creating a DKP cluster on the Google Cloud Platform(GCP).

- GCP Prerequisites (see page 1487)

- GCP Cluster Creation Customization Choices (see page 1494)
- GCP Install in a Non-air-gapped Environment (see page 1500)
- GCP Management Tools (see page 1518)

6.18.1 GCP Prerequisites

6.18.1.1 Prerequisites

Before beginning a DKP installation, verify that you have:

- An x86_64-based Linux or macOS machine with a supported version of the operating system.
- Download (see page 76) the `dkp` binary for Linux, or macOS. To check which version of DKP you installed for compatibility reasons, run the `dkp version -h` command (dkp version (see page 1943)).
- A Container engine/runtime installed is required to bootstrap:
  - Version Docker®[1111] container engine version 18.09.2 or higher installed for Linux or MacOS - On macOS, Docker runs in a virtual machine which needs configured with at least 8 GB of memory.
  - Version 4.0 of Podman[1112] or higher for Linux. Host requirements found here: Host Requirements[1113]
- kubectl[1114] for interacting with the running cluster.
- Install the GCP `gcloud` CLI by following the https://cloud.google.com/sdk/docs/install

6.18.1.1.1 Control plane nodes

You must have at least three control plane nodes. Each control plane node should have at least:

- 4 cores
- 16 GiB memory
- Approximately 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.
- Disk usage must be below 85% on the root volume.

DKP on GCP defaults to deploying an `n2-standard-4` instance with an 80GiB root volume for control plane nodes, which meets the above requirements.

1111 https://docs.docker.com/get-docker/
1112 https://podman.io/getting-started/installation
1113 https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
1114 https://kubernetes.io/docs/tasks/tools/#kubectl

6.18.1.1.2 Worker nodes

You must have at least four worker nodes. The specific number of worker nodes required for your environment can vary depending on the cluster workload and size of the nodes. Each worker node should have at least:

- 8 cores
- 32 GiB memory
- Around 80 GiB of free space for the volume used for `/var/lib/kubelet` and `/var/lib/containerd`.
- Disk usage must be below 85% on the root volume.

DKP on GCP defaults to deploying a `n2-standard-8` instance with an 80GiB root volume for worker nodes, which meets the above requirements.

6.18.1.1.3 GCP Prerequisites:

- Verify that your Google Cloud project does not have the Enable OS Login feature enabled.

The Enable OS Login feature is sometimes enabled by default in GCP projects. If the OS login feature is enabled, KIB will not be able to `ssh` to the VM instances it creates and will not be able to successfully create an image.
To check if it is enabled, use the commands on this page https://cloud.google.com/compute/docs/metadata/setting-custom-metadata#console_2 to inspect the metadata configured in in your project. If you find the the `enable-oslogin` flag set to TRUE, you must remove (or set it to FALSE) to use KIB.

- The user creating the Service Accounts needs additional privileges in addition to the Editor role.
  - See Purple Note Box on GCP Roles (see page 1488)

6.18.1.1.4 Next Topic

GCP Using Konvoy Image Builder (see page 1490)

6.18.1.2 GCP Roles

Service accounts are a special type of Google account that grant permissions to virtual machines instead of end users. Service accounts[1115] are primarily used to ensure safe, managed connections to APIs and Google Cloud services.

These roles are needed when creating an image using Konvoy Image Builder (see page 1646).

1115 https://cloud.google.com/iam/docs/service-account-overview

6.18.1.2.1 GCP Prerequisite Roles

If you are creating your image on either a non-GCP instance or one that does not have the required roles (`Editor` role), you must either:

- Create a GCP service account[1116].
- If you have already created a service account, retrieve the credentials for an existing service account.
- Export the static credentials that will be used to create the cluster:

```
export GCP_B64ENCODED_CREDENTIALS=$(base64 < "$
{GOOGLE_APPLICATION_CREDENTIALS}" | tr -d '\n')
```

> Make sure to rotate static credentials for increased security.

- (option 1) Create a GCP Service Account[1117] using the following `gcloud` commands:

```
export GCP_PROJECT=<your GCP project ID>
export GCP_SERVICE_ACCOUNT_USER=<some new service account user>
export GOOGLE_APPLICATION_CREDENTIALS="$HOME/.gcloud/credentials.json"

gcloud iam service-accounts create "$GCP_SERVICE_ACCOUNT_USER" --
project=$GCP_PROJECT
gcloud projects add-iam-policy-binding $GCP_PROJECT --member="serviceAccount:
$GCP_SERVICE_ACCOUNT_USER@$GCP_PROJECT.iam.gserviceaccount.com" --role=roles/
editor
gcloud iam service-accounts keys create $GOOGLE_APPLICATION_CREDENTIALS --iam-
account="$GCP_SERVICE_ACCOUNT_USER@$GCP_PROJECT.iam.gserviceaccount.com"
```

- (option 2) Retrieve the credentials for an existing service account using the following `gcloud` commands:

```
export GCP_PROJECT=<your GCP project ID>
export GCP_SERVICE_ACCOUNT_USER=<existing service account user>
export GOOGLE_APPLICATION_CREDENTIALS="$HOME/.gcloud/credentials.json"

gcloud iam service-accounts keys create $GOOGLE_APPLICATION_CREDENTIALS --iam-
account="$GCP_SERVICE_ACCOUNT_USER@$GCP_PROJECT.iam.gserviceaccount.com"
```

- Export the static credentials that will be used to create the cluster:

1116 https://cloud.google.com/iam/docs/service-account-overview
1117 https://cloud.google.com/iam/docs/service-account-overview

```
export GCP_B64ENCODED_CREDENTIALS=$(base64 < "$
{GOOGLE_APPLICATION_CREDENTIALS}" | tr -d '\n')
```

Refer to DKP Documentation regarding roles and minimum permission for use with Konvoy Image Builder: GCP Roles (see page 1488)

To create a **GCP Service Account** with the `Editor` role, the user creating the GCP Service Account needs the `Editor`, `RoleAdministrator`, and `SecurityAdmin` roles. However, those pre-defined roles grant more permissions than the minimum set needed to create a DKP cluster.

For **DKP** cluster creation**, a minimal** set of roles and permissions needed for the user creating the GCP Service Account is the `Editor` role ***plus*** the following additional permissions:

- `compute.disks.setIamPolicy`
- `compute.instances.setIamPolicy`
- `iam.roles.create`
- `iam.roles.delete`
- `iam.roles.update`
- `iam.serviceAccounts.setIamPolicy`
- `resourcemanager.projects.setIamPolicy`

For more information on GCP service accounts, see GCP's documentation:

- https://cloud.google.com/iam/docs/creating-managing-service-accounts
- https://cloud.google.com/iam/docs/best-practices-service-accounts

6.18.1.2.1.1 Next Topic

GCP Using Konvoy Image Builder (see page 1490)

6.18.1.3 GCP Using Konvoy Image Builder

Konvoy Image Builder (KIB) is a complete solution for building Cluster API compliant images.

This procedure describes how to use the Konvoy Image Builder (see page 1626) (KIB) to create a Cluster API[1118] compliant GCP image. GCP images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create a GCP image of your current computer system settings and software. The GCP image can then be replicated and distributed, creating your computer system for other users. KIB uses variable overrides (see page 1670) to specify base image and container images to use in your new GCP image.

Google Cloud Platform does not publish images. You must first build the image using Konvoy Image Builder. Explore the Customize your Image (see page 1661) topic for more options. For more information regarding using the image in creating clusters, refer to the GCP Infrastructure (see page 1486) section of the documentation.

6.18.1.3.1 DKP Prerequisites

Before you begin, you must:

- Download the KIB (see page 0) bundle for your version of DKP.
- Check the Supported Infrastructure Operating Systems (see page 67)
- Check the Supported Kubernetes Version (see page 1706) for your Provider.
- Create a working Docker or other registry setup.
- On Debian-based Linux distributions, install a version of the cri-tools[1119] package known to be compatible with both the Kubernetes and container runtime versions.
- Verify that your Google Cloud project does not have the Enable OS Login feature enabled. See below for more information:

The Enable OS Login feature is sometimes enabled by default in GCP projects. If the OS login feature is enabled, KIB will not be able to `ssh` to the VM instances it creates and will not be able to successfully create an image.
To check if it is enabled, use the commands on this page https://cloud.google.com/compute/docs/metadata/setting-custom-metadata#console_2 to inspect the metadata configured in in your project. If you find the the `enable-oslogin` flag set to TRUE, you must remove (or set it to FALSE) to successfully use KIB.

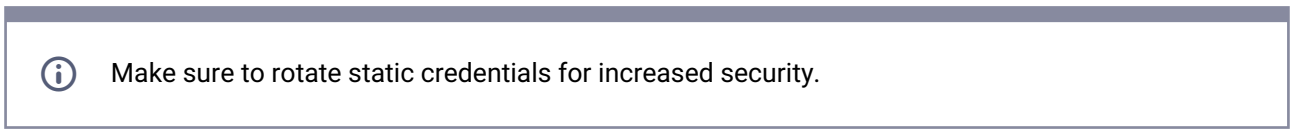
6.18.1.3.2 GCP Prerequisite Roles

If you are creating your image on either a non-GCP instance or one that does not have the required roles (see page 1488), you must either:

1118 https://cluster-api.sigs.k8s.io/
1119 https://github.com/kubernetes-sigs/cri-tools

- Create a GCP service account[1120].
- If you have already created a service account, retrieve the credentials for an existing service account.
- Export the static credentials that will be used to create the cluster:

```
export GCP_B64ENCODED_CREDENTIALS=$(base64 < "$
{GOOGLE_APPLICATION_CREDENTIALS}" | tr -d '\n')
```



Make sure to rotate static credentials for increased security.

6.18.1.3.3 Build the GCP image

1. Run the `konvoy-image` command to build and validate the image:

```
./konvoy-image build gcp --project-id ${GCP_PROJECT} --network ${NETWORK_NAME}
images/gcp/ubuntu-2004.yaml
```

2. KIB will run and print out the name of the created image, you will use this name when creating a Kubernetes cluster. See sample output below:

```
...
==> ubuntu-2004-focal-v20220419: Deleting instance...
    ubuntu-2004-focal-v20220419: Instance has been deleted!
==> ubuntu-2004-focal-v20220419: Creating image...
==> ubuntu-2004-focal-v20220419: Deleting disk...
    ubuntu-2004-focal-v20220419: Disk has been deleted!
==> ubuntu-2004-focal-v20220419: Running post-processor: manifest
Build 'ubuntu-2004-focal-v20220419' finished after 7 minutes 46 seconds.

==> Wait completed after 7 minutes 46 seconds

==> Builds finished. The artifacts of successful builds are:
--> ubuntu-2004-focal-v20220419: A disk image was created: konvoy-ubuntu-2004-1-2
3-7-1658523168
--> ubuntu-2004-focal-v20220419: A disk image was created: konvoy-ubuntu-2004-1-2
3-7-1658523168
```

1120 https://cloud.google.com/iam/docs/service-account-overview

Ensure you have named the correct YAML file for your OS[1121] in the `konvoy-image build` command.

3. To find a list of images you have created in your account, run the following command:

```
gcloud compute images list --no-standard-images
```

6.18.1.3.4 Create a Network (optional)

Building an image requires a Network[1122] with firewall rules that allow SSH access to the VM instance.

1. Set your GCP Project ID for your `gcp` account unless already set previously:

```
export GCP_PROJECT=<your GCP project ID>
```

2. Run the following to create a new network:

```
export NETWORK_NAME=kib-ssh-network
gcloud compute networks create "${NETWORK_NAME}" --project="${GCP_PROJECT}" --
subnet-mode=auto --mtu=1460 --bgp-routing-mode=regional
```

3. Create the firewall rule to allow Ingress access on port 22:

```
gcloud compute firewall-rules create "${NETWORK_NAME}-allow-ssh" --project="$
{GCP_PROJECT}" --network="projects/${GCP_PROJECT}/global/networks/$
{NETWORK_NAME}" --description="Allows TCP connections from any source to any
instance on the network using port 22." --direction=INGRESS --priority=65534 --
source-ranges=0.0.0.0/0 --action=ALLOW --rules=tcp:22
```

With your KIB image now created, you can now move on to create your cluster and set up your Cluster API[1123] (CAPI) controllers.

1121 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/gcp
1122 https://cloud.google.com/vpc/docs/vpc
1123 https://cluster-api.sigs.k8s.io/

Konvoy Image Builder (see page 1626) has a more detailed section in the documentation if you need to refer there for compatible versions with DKP and other specific information.

6.18.1.3.4.1 Next Topic

GCP Cluster Creation Customization Choices (see page 1494)

6.18.2 GCP Cluster Creation Customization Choices

Below are a some methods to customize your cluster during creation. If none of these choices apply, skip this page and proceed to:

- GCP Install in a Non-air-gapped Environment (see page 1500)

6.18.2.1 Section Topics

When creating clusters, many options are available such as those listed in this section of the documentation. Familiarize yourself with the flags required to apply these customizations during cluster creation.

- GCP Customizing CAPI Clusters (see page 1494) — Familiarize yourself with Cluster API before editing the cluster objects because edits can prevent the cluster from deploying successfully.
- GCP Registry Mirrors (see page 1495) — Configure your cluster to use an existing local registry when attempting to pull images by adding the flag(s) to the dkp create cluster command to pull images from your local registry.
- GCP Load the Registry (see page 1496) — Because air-gapped environments do not have direct access to the Internet, you must download, extract and load several required images to your local container registry https://d2iq.atlassian.net/wiki/spaces/DENT/pages/210829371, before installing DKP.
- GCP HTTP Proxy (see page 1499) — When creating a DKP cluster in environments that use an HTTP/HTTPS proxy, you must provide proxy details. The proxy values are strings that list a set of proxy servers, URLs, or wildcard addresses that is specific to your environment.
- GCP Output Directory YAML (see page 1500) — You can create individual files with different smaller manifests for ease in editing using the --output-directory flag used with --output=json|yaml. You create the directory of where to output resources to files.

6.18.2.1.1 Next Step:

After you make decisions about customization choices, proceed to GCP environment instructions:

- GCP Install in a Non-air-gapped Environment (see page 1500)

6.18.2.2 GCP Customizing CAPI Clusters

Familiarize yourself with Cluster API (see page 1061) before editing the cluster objects because edits can prevent the cluster from deploying successfully.

The result of this command will allow such edits:

```
dkp create cluster gcp \
    --cluster-name=${CLUSTER_NAME} \
    --dry-run \
    --output=yaml \
  > ${CLUSTER_NAME}.yaml
```

To edit the YAML, you need to understand the CAPI components to avoid breaking the cluster. For expanded component explanation, refer to the Universal Configurations for All Providers section of the documentation in the section: Customizing CAPI Components for a Cluster (see page 1061)

6.18.2.2.1 Next Topic

GCP Registry Mirrors (see page 1495)

6.18.2.3 GCP Registry Mirrors

Configure your cluster to use an existing local registry (see page 1606) when attempting to pull images by adding the flag(s) to the `dkp create cluster` command to pull images from your local registry.

Kubernetes does not natively provide a registry for hosting the container images you will use to run the applications you want to deploy on Kubernetes. Instead, Kubernetes requires you to use an external solution for storing and sharing container images. There are a variety of Kubernetes-compatible registry options that are compatible with DKP.

6.18.2.3.1 How Does it Work?

The **first time** you request an image from your local registry mirror, it pulls the image from the public registry (such as Docker) and stores it locally before handing it back to you. On subsequent requests, the local registry mirror is able to serve the image from its own storage.

6.18.2.3.2 Air-gapped vs Non-air-gapped Environments

In a non-air-gapped environment, you have access to the Internet. You retrieve artifacts from specialized repositories dedicated to them, such as Docker images contained in DockerHub and Helm Charts that come from a dedicated Helm Chart repository. You can also create your own local repository to hold the downloaded container images needed or any custom images you've created with the Konvoy Image Builder (see page 1626) tool.

In an **air-gapped environment**, you need a local repository to store Helm charts, Docker images, and other artifacts. Private registries provide security and privacy into enterprise container image storage, whether hosted remotely or on-premises locally in an air-gapped environment. DKP in an air-gapped environment **requires** a local container registry of trusted images to enable production-level Kubernetes cluster management. However, a local registry is an option in a non-air-gapped environment as well for speed and security.

If you want to use images from this local registry to deploy applications inside your Kubernetes cluster, you'll need to set up a **secret** for a private registry. The secret contains your login data, which Kubernetes needs to connect to your private repository.

6.18.2.3.2.1 Related Topics

More information and detail can be found:

- Registry Mirror Tools (see page 1606)
- Use a Registry Mirror (see page 1609)

6.18.2.3.2.2 Next Step

GCP Load the Registry (see page 1496)

6.18.2.4 GCP Load the Registry

Because air-gapped environments do not have direct access to the Internet, you must download, extract and load several required images to your local container registry (see page 1606), before installing DKP.

If desired, environments that are non-air-gapped can also perform the follow steps to use a local registry for speed and security reasons.

6.18.2.4.1 Load Images into your Registry

This page assumes you have already downloaded (see page 76) and extracted (see page 1211) the bundle from the Prerequisites (see page 1151). The sections below explain further how you are pushing the images to your registry and then using them in creating a cluster.

6.18.2.4.2 Download all Images for Air-gapped Deployments

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for prerequisites to download and then how to push the necessary images to this registry.

1. Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`) to load registry images as explained below.
2. Connectivity with clusters attaching to the management cluster is required:
    - Both management and attached clusters must be able to connect to the local registry.
    - The management cluster must be able to connect to all attached cluster's API servers.
    - The management cluster must be able to connect to any load balancers created for platform services on the management cluster.

6.18.2.4.3 Extract Air-gapped Images and Set Variables

Follow these steps to **extract** the air-gapped image bundles into your private registry:

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

   ```
   cd dkp-v2.8.1
   ```

3. Set an environment variable with your registry address:

   ```
   export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   ```

- `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- NOTE: Other local registries may use the options below:
  - JFrog - `CONTAINER_REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
  - `CONTAINER_REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
  - `CONTAINER_REGISTRY_PASSWORD` : optional if username is not set.

6.18.2.4.3.1 Load Images to your Private Registry - Konvoy

Before creating or upgrading a Kubernetes cluster, you need to load the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, refer to Local Registry Tools (see page 1606) page for more information.

Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

6.18.2.4.3.2 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

6.18.2.4.3.3 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME}
--to-registry-password=${REGISTRY_PASSWORD}
```

For more information to set up a private registry with a registry mirror, see this page (see page 1609) for details on using that flag.

6.18.2.4.3.4 Next Topic

GCP HTTP Proxy (see page 1499)

6.18.2.5 GCP HTTP Proxy

When creating a DKP cluster in environments that use an HTTP/HTTPS proxy, you must provide proxy details. The proxy values are strings that list a set of proxy servers, URLs, or wildcard addresses that is specific to your environment.

If your environment uses HTTP/HTTPS proxies, you must include the flags and their related values in commands for the proxy to be successful throughout various steps of installation:

- `--http-proxy`
- `--https-proxy`
- `--no-proxy`

Create the bootstrap cluster and CAPI components using the appropriate commands, `dkp create bootstrap` and `dkp create capi-components` respectively, combined with the command line flags to include your HTTP/S proxy information.

You can also specify HTTP/S proxy information in an override file when using Konvoy Image Builder (KIB) (see page 1626).

Without these values provided as part of the relevant `dkp create` command, DKP cannot create the requisite parts of your new cluster correctly. This is true of both management and managed clusters (see page 79) alike.

To create a proxied environment, you need to include flags at these action item points:

- Bootstrap cluster
- CAPI components
- Cluster creation
- DKP Kommander component

For full HTTP Proxy configuration, you need to specify proxy settings using all the details in the Configuring an HTTP/HTTPS Proxy (see page 1053)section of the documentation for:

- Configure the Bootstrap Cluster HTTP Proxy Settings (see page 1054)
- Create CAPI Components with HTTP/HTTPS Proxy Settings (see page 1055)
- Create a Cluster with HTTP/HTTPS Proxy (see page 1056)
- Configure an HTTP/HTTPS Proxy for the DKP Kommander Component (see page 1058)

6.18.2.5.1 HTTP Proxy Cluster Example

```
dkp create cluster gcp \
 --cluster-name ${CLUSTER_NAME} \
 --control-plane-http-proxy="${CONTROL_PLANE_HTTP_PROXY}" \
```

```
  --control-plane-https-proxy="${CONTROL_PLANE_HTTPS_PROXY}" \
  --control-plane-no-proxy="${CONTROL_PLANE_NO_PROXY}" \
  --worker-http-proxy="${WORKER_HTTP_PROXY}" \
  --worker-https-proxy="${WORKER_HTTPS_PROXY}" \
  --worker-no-proxy="${WORKER_NO_PROXY}"
```

6.18.2.5.2 Next Topic

GCP Output Directory YAML (see page 1500)

6.18.2.6 GCP Output Directory YAML

You can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag used with `--output=json|yaml`. You create the directory of where to output resources to files.

Using this flag will create multiple files in the specified directory which must already exist:

```
dkp create cluster gcp
    --cluster-name=${CLUSTER_NAME} \
    --dry-run \
    --output=yaml \
    --output-directory=<existing-directory>
```

For more information regarding this flag or others, please refer to the CLI section of the documentation for the dkp create cluster (see page 1853) command and select your provider.

6.18.2.6.1 Next Step

If no customizations are desired, continue to GCP Install in a Non-air-gapped Environment (see page 1500).

6.18.3 GCP Install in a Non-air-gapped Environment

This section provides instructions to install DKP in a GCP non-air-gapped environment with custom settings. First you create an Bootstrap Cluster (see page 1187) and then move the CAPI resources to the workload cluster and delete the bootstrap cluster.

If not already done, refer to Get Started (see page 77) section of the documentation for:

- Resource Requirements (see page 119)
- Install Overview (see page 127)
- Prerequisites for Install (see page 141)

6.18.3.1 GCP Specific Prerequisites

Before you begin using Konvoy with GCP, you must:

- Verify that your Google Cloud project does not have the Enable OS Login feature enabled.

> The Enable OS Login feature is sometimes enabled by default in GCP projects. If the OS login feature is enabled, KIB will not be able to `ssh` to the VM instances it creates and will not be able to successfully create an image.
> To check if it is enabled, use the commands on this page https://cloud.google.com/compute/docs/metadata/setting-custom-metadata#console_2 to inspect the metadata configured in in your project. If you find the the `enable-oslogin` flag set to TRUE, you must remove (or set it to FALSE) to use KIB.

- The user creating the Service Accounts needs additional privileges in addition to the Editor role.
  - See GCP Prerequisite Roles (see page 1488)

> To deploy a cluster with a custom image in a region where CAPI images[1124] are not provided, you need to use Konvoy Image Builder (see page 1153) to create your own image for the region.

6.18.3.1.1 Next Step

Bootstrap GCP (see page 1501)

6.18.3.2 Bootstrap GCP

To create Kubernetes clusters, DKP uses Cluster API[1125] (CAPI) controllers. These controllers run on a Kubernetes cluster. To get started, you need a *bootstrap* cluster. By default, DKP creates a bootstrap cluster for you in a Docker container using the Kubernetes-in-Docker (KIND[1126]) tool.

6.18.3.2.1 Prerequisites

Before you begin, you must:

1124 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html
1125 https://cluster-api.sigs.k8s.io/
1126 https://github.com/kubernetes-sigs/kind

- Complete the steps in Prerequisites (see page 1487).
- Ensure the `dkp` binary can be found in your `$PATH`.

6.18.3.2.1.1 Bootstrap Cluster Lifecycle Services

1. Review Universal Configurations for all Infrastructure Providers (see page 1052) regarding settings, flags and other choices and then begin bootstrapping.
2. Create a bootstrap cluster:

   ```
   dkp create bootstrap --kubeconfig $HOME/.kube/config
   ```

   - Configuring an HTTP/HTTPS Proxy (see page 1053) use `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful.

HTTP only flags if needed:

To create a bootstrap cluster in a proxied environment (see page 1054) use this command syntax, in addition to any other flags you may need:

```
dkp create bootstrap --kubeconfig $HOME/.kube/config \
  --http-proxy <string> \
  --https-proxy <string> \
  --no-proxy <string>
```

3. The output looks similar to:

   ```
   ✓ Creating a bootstrap cluster
   ✓ Initializing new CAPI components
   ```

4. DKP creates a bootstrap cluster using KIND[1127] as a library. DKP then deploys the following Cluster API[1128] providers on the cluster:
   - Core Provider[1129]
   - GCP Infrastructure Provider[1130]
   - Kubeadm Bootstrap Provider[1131]
   - Kubeadm ControlPlane Provider[1132]

1127 https://github.com/kubernetes-sigs/kind
1128 https://cluster-api.sigs.k8s.io/
1129 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/
1130 https://github.com/kubernetes-sigs/cluster-api-provider-gcp
1131 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/bootstrap/kubeadm
1132 https://github.com/kubernetes-sigs/cluster-api/tree/v0.3.20/controlplane/kubeadm

5. DKP waits until the controller-manager and webhook deployments of these providers are ready. List these deployments using this command:

```
kubectl get --all-namespaces deployments -l=clusterctl.cluster.x-k8s.io
```

You will then receive the following output:

```
NAMESPACE                           NAME
READY   UP-TO-DATE   AVAILABLE   AGE
capa-system                         capa-controller-manager
1/1     1            1           1h
capg-system                         capg-controller-manager
1/1     1            1           1h
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-manager
1/1     1            1           1h
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-
manager   1/1     1            1           1h
capi-system                         capi-controller-manager
1/1     1            1           1h
cappp-system                        cappp-controller-manager
1/1     1            1           1h
capv-system                         capv-controller-manager
1/1     1            1           1h
capz-system                         capz-controller-manager
1/1     1            1           1h
cert-manager                        cert-manager
1/1     1            1           1h
cert-manager                        cert-manager-cainjector
1/1     1            1           1h
cert-manager                        cert-manager-webhook
1/1     1            1           1h
```

6.18.3.2.1.2 Next Step

Create a New GCP Cluster (see page 1503)

6.18.3.3 Create a New GCP Cluster

6.18.3.3.1 Name your Cluster

1. Give your cluster a unique name suitable for your environment. The cluster name may only contain the following characters: `a-z`, `0-9`, `.`, and `-`. Cluster creation will fail if the name has capital letters. See Kubernetes[1133] for more naming information.

1133 https://kubernetes.io/docs/concepts/overview/working-with-objects/names/

In GCP it is critical that the name is unique, as no two clusters in the same GCP account can have the same name.

2. Set the environment variable:

```
export CLUSTER_NAME=<gcp-example>
```

To increase Docker Hub's rate limit[1134] use your Docker Hub credentials when creating the cluster, by setting the following flag `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username=<username> --registry-mirror-password=<password>` on the `dkp create cluster` command.

6.18.3.3.2 Create a New GCP Cluster

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools is to ensure your cluster has nodes deployed in multiple Availability Zones.

By default, the control-plane Nodes will be created in 3 different zones. However, the default worker Nodes will reside in a single zone. You may create additional node pools in other zones with the `dkp create nodepool` command. The default region for the availability zones is `us-west1`.

Google Cloud Platform does not publish images. You must first build the image using Konvoy Image Builder (see page 1626).

1. Create an image using Konvoy Image Builder (KIB) (see page 1626) and then export the image name:

```
export IMAGE_NAME=projects/${GCP_PROJECT}/global/images/<image_name_from_kib>
```

1134 https://docs.docker.com/docker-hub/download-rate-limit/

2. Ensure your subnets (see page 1065) do not overlap with your host subnet because they cannot be changed after cluster creation. If you need to change the kubernetes subnets, you must do this at cluster creation. The default subnets used in DKP are:

```
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    services:
      cidrBlocks:
      - 10.96.0.0/12
```

3. **(Optional)** Modify Control Plane Audit logs - Users can make modifications to the `KubeadmControlplane` cluster-api object to configure different `kubelet` options. See the following guide (see page 1613) if you wish to configure your control plane beyond the existing options that are available from flags.
4. **(Optional)** Determine what VPC Network to use. All GCP accounts come with a preconfigured VPC Network named `default`, which will be used if you do not specify a different network.
   To use a different VPC network for your cluster, create one by following these instructions for Create and Manage VPC Networks[1135]. Then specify the `--network <new_vpc_network_name>` option on the create cluster command below. More information is available on GCP Cloud Nat[1136] and network flag.
5. **(Optional)** Use a registry mirror (see page 1609). Configure your cluster to use an existing local registry (see page 1606) as a mirror when attempting to pull images previously pushed to your registry (see page 317).

Export Registry Variables

If you have a local registry, you must provide additional arguments when creating the cluster. These tell the cluster where to locate the local registry to use by defining the URL. Set the environment variable with your registry information:

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_CA=<path to the CA on the bastion>
export REGISTRY_USERNAME=<username>
export REGISTRY_USERNAME=<password>
```

- `REGISTRY_URL`: the address of an existing container registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.

1135 https://cloud.google.com/vpc/docs/create-modify-vpc-networks#create-auto-network

1136 https://github.com/kubernetes-sigs/cluster-api-provider-gcp/blob/3406adaae0f8f65a615844e55d856d306eddacc1/docs/book/src/topics/prerequisites.md#cloud-nat

- `REGISTRY_CA` : (optional) the path on the bastion machine to the container registry CA. Konvoy will configure the cluster nodes to trust this CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- `REGISTRY_USERNAME` : optional, set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.

Set the flag during cluster creation:

```
--registry-mirror-url=${REGISTRY_URL} \
--registry-mirror-cacert=${REGISTRY_CA} \
--registry-mirror-username=${REGISTRY_USERNAME} \
--registry-mirror-password=${REGISTRY_PASSWORD}
```

See also: GCP Registry Mirrors (see page 1495)

6. Create a Kubernetes cluster. The following example shows a common configuration. See dkp create cluster gcp (see page 1869) reference for the full list of cluster creation options.

   ```
   dkp create cluster gcp \
   --cluster-name=${CLUSTER_NAME} \
   --additional-tags=owner=$(whoami) \
   --with-gcp-bootstrap-credentials=true \
   --project=${GCP_PROJECT} \
   --image=${IMAGE_NAME} \
   --dry-run \
   --output=yaml \
   > ${CLUSTER_NAME}.yaml
   ```

Expand the drop-downs for **HTTP and other flags** to use during the cluster creation step above. For more information regarding this flag or others, please refer to the CLI section of the documentation for dkp create cluster (see page 1853) and select your provider.

HTTP Only:

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy` , `--https-proxy` , and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

```
--http-proxy <<http proxy list>>
--https-proxy <<https proxy list>>
--no-proxy <<no proxy list>>
```

- To configure the Control Plane and Worker nodes to use an HTTP proxy:

```
export CONTROL_PLANE_HTTP_PROXY=http://example.org:8080
export CONTROL_PLANE_HTTPS_PROXY=http://example.org:8080
export
CONTROL_PLANE_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1
,10.96.0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.s
vc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.sv
c.cluster,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"

export WORKER_HTTP_PROXY=http://example.org:8080
export WORKER_HTTPS_PROXY=http://example.org:8080
export
WORKER_NO_PROXY="example.org,example.com,example.net,localhost,127.0.0.1,10.96.
0.0/12,192.168.0.0/16,kubernetes,kubernetes.default,kubernetes.default.svc,kube
rnetes.default.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.clust
er,.svc.cluster.local,169.254.169.254,.elb.amazonaws.com"
```

- Replace:
  - `example.org,example.com,example.net` with you internal addresses
  - `localhost` and `127.0.0.1` addresses should not use the proxy
  - `10.96.0.0/12` is the default Kubernetes service subnet
  - `192.168.0.0/16` is the default Kubernetes pod subnet
  - `kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cluster,kubernetes.default.svc.cluster.local` is the internal Kubernetes kube-apiserver service
  - `.svc,.svc.cluster,.svc.cluster.local` is the internal Kubernetes services
  - `169.254.169.254` is the AWS metadata server
  - `.elb.amazonaws.com` is for the worker nodes to allow them to communicate directly to the kube-apiserver ELB
- Create a Kubernetes cluster with HTTP proxy configured using these flags during `dkp create cluster` command:

```
--control-plane-http-proxy=${CONTROL_PLANE_HTTP_PROXY} \
--control-plane-https-proxy=${CONTROL_PLANE_HTTPS_PROXY} \
--control-plane-no-proxy=${CONTROL_PLANE_NO_PROXY} \
--worker-http-proxy=${WORKER_HTTP_PROXY} \
--worker-https-proxy=${WORKER_HTTPS_PROXY} \
--worker-no-proxy=${WORKER_NO_PROXY} \
```

Individual manifests using the Output Directory flag:

You can create individual files with different smaller manifests for ease in editing using the `--output-directory` flag. This will create multiple files in the specified directory which must already exist:

```
--output-directory=<existing-directory>
```

Refer to the Cluster Creation Customization Choices (see page 1170) section for more information on how to use optional flags such as the `--output-directory` flag.

7. Inspect or edit the cluster objects and familiarize yourself with Cluster API before editing the cluster objects as edits can prevent the cluster from deploying successfully. See GCP Customizing CAPI Clusters (see page 1494)
8. **(Optional)** Modify Control Plane Audit logs - Users can make modifications to the `KubeadmControlplane` cluster-api object to configure different `kubelet` options. See the following guide (see page 1613) if you wish to configure your control plane beyond the existing options that are available from flags.
9. Create the cluster from the objects generated from the `dry run`. A warning will appear in the console if the resource already exists and will require you to remove the resource or update your YAML.

   ```
   kubectl create -f ${CLUSTER_NAME}.yaml
   ```

   NOTE: If you used the `--output-directory` flag in your `dkp create .. --dry-run` step above, create the cluster from the objects you created by specifying the directory:

   ```
   kubectl create -f <existing-directory>/.
   ```

10. Wait for the cluster control-plane to be ready:

    ```
    kubectl wait --for=condition=ControlPlaneReady "clusters/${CLUSTER_NAME}" --
    timeout=20m
    ```

11. After the objects are created on the API server, the Cluster API controllers reconcile them. They create infrastructure and machines. As they progress, they update the Status of each object. Konvoy provides a command to describe the current status of the cluster:

    ```
    dkp describe cluster -c ${CLUSTER_NAME}
    ```

Output:

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/gcp-example                                                 True
52s
├─ClusterInfrastructure - GCPCluster/gcp-example
```

```
├─ControlPlane - KubeadmControlPlane/gcp-example-control-plane            True
52s
│ ├─Machine/gcp-example-control-plane-6fbzn                               True
2m32s
│ │ └─MachineInfrastructure - GCPMachine/gcp-example-control-plane-62g6s
│ ├─Machine/gcp-example-control-plane-jf6s2                               True
7m36s
│ │ └─MachineInfrastructure - GCPMachine/gcp-example-control-plane-bsr2z
│ └─Machine/gcp-example-control-plane-mnbfs                               True
54s
│   └─MachineInfrastructure - GCPMachine/gcp-example-control-plane-s8xsx
└─Workers
  └─MachineDeployment/gcp-example-md-0                                    True
78s
    ├─Machine/gcp-example-md-0-68b86fddb8-8glsw                           True
2m49s
    │ └─MachineInfrastructure - GCPMachine/gcp-example-md-0-zls8d
    ├─Machine/gcp-example-md-0-68b86fddb8-bvbm7                           True
2m48s
    │ └─MachineInfrastructure - GCPMachine/gcp-example-md-0-5zcvc
    ├─Machine/gcp-example-md-0-68b86fddb8-k9499                           True
2m49s
    │ └─MachineInfrastructure - GCPMachine/gcp-example-md-0-k8h5p
    └─Machine/gcp-example-md-0-68b86fddb8-l6vfb                           True
2m49s
      └─MachineInfrastructure - GCPMachine/gcp-example-md-0-9h5vn
```

DKP uses the GCP CSI driver as the default storage provider (see page 110). Use a Kubernetes CSI[1137] compatible storage that is suitable for production. See the Kubernetes documentation called Changing the Default Storage Class[1138] for more information.

If you're not using the default, you cannot deploy an alternate provider until ***after*** the `dkp create cluster` is finished. However, it must be determined before Kommander installation.

6.18.3.3.2.1 Next Step

Make the New GCP Cluster Self-Managed (see page 1509)

6.18.3.4 Make the New GCP Cluster Self-Managed

DKP deploys all cluster lifecycle services to a bootstrap cluster, which then deploys a workload cluster. When the workload cluster is ready, move the cluster lifecycle services to the workload cluster, which makes the workload cluster self-managed. This section describes how to make a workload cluster self-managed.

1137 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

1138 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/

Before starting, ensure you create a workload cluster as described in Create a New GCP Cluster (see page 1503) .

This page contains instructions on how to make your cluster self-managed. This is necessary if there is only one cluster in your environment, or if this cluster should become the Management cluster in a multi-cluster environment.

If you already have a self-managed or Management cluster in your environment, skip this page.

6.18.3.4.1 Make the New Kubernetes Cluster Manage Itself

Follow these steps to turn your new cluster into a **Management Cluster** for Enterprise (or a free-standing Essential Cluster):

1. Deploy cluster lifecycle services on the workload cluster:

   ```
   dkp create capi-components --kubeconfig ${CLUSTER_NAME}.conf
   ```

   Output:

   ```
   ✓ Initializing new CAPI components
   ```

   If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

2. Move the Cluster API objects from the bootstrap to the workload cluster:

   The cluster lifecycle services on the workload cluster are ready, but the workload cluster configuration is on the bootstrap cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the bootstrap to the workload cluster. This process is also called a Pivot[1139].

   ```
   dkp move capi-resources --to-kubeconfig ${CLUSTER_NAME}.conf
   ```

   Output:

1139 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
✓ Moving cluster resources
You can now view resources in the moved cluster by using the --kubeconfig flag
with kubectl. For example: kubectl --kubeconfig=gcp-example.conf get nodes
```

To ensure only one set of cluster lifecycle services manages the workload cluster, DKP first pauses reconciliation of the objects on the bootstrap cluster, then creates the objects on the workload cluster. As DKP copies the objects, the cluster lifecycle services on the workload cluster reconcile the objects. The workload cluster becomes self-managed after DKP creates all the objects. If it fails, the `move` command can be safely retried.

3. Wait for the cluster control-plane to be ready:

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf wait --for=condition=ControlPlaneRead
y "clusters/${CLUSTER_NAME}" --timeout=20m
```

Output:

```
cluster.cluster.x-k8s.io/gcp-example condition met
```

4. Use the cluster lifecycle services on the workload cluster to check the workload cluster status. After moving the cluster lifecycle services to the workload cluster, remember to use DKP with the workload cluster `kubeconfig`.

```
dkp describe cluster --kubeconfig ${CLUSTER_NAME}.conf -c ${CLUSTER_NAME}
```

Output:

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/gcp-example                                                 True
14s
├─ClusterInfrastructure - GCPCluster/gcp-example
├─ControlPlane - KubeadmControlPlane/gcp-example-control-plane      True
14s
| ├─Machine/gcp-example-control-plane-6fbzn                         True
17s
| | └─MachineInfrastructure - GCPMachine/gcp-example-control-plane-62g6s
| ├─Machine/gcp-example-control-plane-jf6s2                         True
17s
| | └─MachineInfrastructure - GCPMachine/gcp-example-control-plane-bsr2z
```

```
| └─Machine/gcp-example-control-plane-mnbfs                                  True
17s
|   └─MachineInfrastructure - GCPMachine/gcp-example-control-plane-s8xsx
└─Workers
  └─MachineDeployment/gcp-example-md-0                                       True
17s
    ├─Machine/gcp-example-md-0-68b86fddb8-8glsw                              True
17s
    | └─MachineInfrastructure - GCPMachine/gcp-example-md-0-zls8d
    ├─Machine/gcp-example-md-0-68b86fddb8-bvbm7                              True
17s
    | └─MachineInfrastructure - GCPMachine/gcp-example-md-0-5zcvc
    ├─Machine/gcp-example-md-0-68b86fddb8-k9499                              True
17s
    | └─MachineInfrastructure - GCPMachine/gcp-example-md-0-k8h5p
    └─Machine/gcp-example-md-0-68b86fddb8-l6vfb                              True
17s
      └─MachineInfrastructure - GCPMachine/gcp-example-md-0-9h5vn
```

5. Remove the bootstrap cluster, as the workload cluster is now self-managed:

```
dkp delete bootstrap --kubeconfig $HOME/.kube/config
✓ Deleting bootstrap cluster
```

6.18.3.4.2 Known Limitations

DKP only supports moving all namespaces in the cluster; DKP does not support migration of individual namespaces.

6.18.3.4.3 Next Step

Explore the GCP Cluster (see page 1512)

6.18.3.5 Explore the GCP Cluster

This guide explains how to use the command line to interact with your newly deployed Kubernetes cluster. Before you start, make sure you have created a workload cluster, as described in Create a New GCP Cluster (see page 1503).

6.18.3.5.1 Explore the New Kubernetes Cluster

1. Get a `kubeconfig` file for the workload cluster:

   When the workload cluster is created, the cluster lifecycle services generate a kubeconfig file for the workload cluster, and write it to a *Secret*. The kubeconfig file is scoped to the cluster administrator.

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

2. Verify the API server is up by listing the nodes:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get nodes
```

NOTE: It may take a few minutes for the Status to move to Ready while the Pod network is deployed. The Nodes' Status should change to Ready soon after the calico-node DaemonSet Pods are Ready. You should receive the following output:

```
NAME                                STATUS   ROLES                  AGE
VERSION
gcp-example-control-plane-9z77w   Ready    control-plane,master   4m44s
v1.27.6
gcp-example-control-plane-rtj9h   Ready    control-plane,master   104s
v1.27.6
gcp-example-control-plane-zbf9w   Ready    control-plane,master   3m23s
v1.27.6
gcp-example-md-0-88c46            Ready    <none>                 3m28s   v1.27
.6
gcp-example-md-0-fp8s7            Ready    <none>                 3m28s   v1.27
.6
gcp-example-md-0-qvnx7            Ready    <none>                 3m28s   v1.27
.6
gcp-example-md-0-wjdrg            Ready    <none>                 3m27s   v1.27
.6
```

3. List the Pods with the command:

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf get pods -A
```

4. View the following output:

```
NAMESPACE                           NAME
READY   STATUS    RESTARTS       AGE
calico-system                       calico-kube-controllers-577c696df9-v2nzv
1/1     Running   0              5m23s
calico-system                       calico-node-4x5rk
1/1     Running   0              4m22s
calico-system                       calico-node-cxsgc
1/1     Running   0              4m23s
calico-system                       calico-node-dvlnm
1/1     Running   0              4m23s
calico-system                       calico-node-h6nlt
1/1     Running   0              4m23s
```

```
calico-system                       calico-node-jmkwq
1/1     Running   0             5m23s
calico-system                       calico-node-tnf54
1/1     Running   0             4m18s
calico-system                       calico-node-v6bwq
1/1     Running   0             2m39s
calico-system                       calico-typha-6d8c94bfdf-dkfvq
1/1     Running   0             5m23s
calico-system                       calico-typha-6d8c94bfdf-fdfn2
1/1     Running   0             3m43s
calico-system                       calico-typha-6d8c94bfdf-kjgzj
1/1     Running   0             3m43s
capa-system                         capa-controller-manager-6468bc488-w7nj9
1/1     Running   0             67s
capg-system                         capg-controller-manager-5fb47f869b-6jgms
1/1     Running   0             53s
capi-kubeadm-bootstrap-system       capi-kubeadm-bootstrap-controller-
manager-65ffc94457-7cjdn      1/1     Running   0             74s
capi-kubeadm-control-plane-system   capi-kubeadm-control-plane-controller-
manager-bc7b688d4-vv8wg   1/1     Running   0             72s
capi-system                         capi-controller-manager-dbfc7b49-dzvw8
1/1     Running   0             77s
cappp-system                        cappp-controller-manager-8444d67568-rmms2
1/1     Running   0             59s
capv-system                         capv-controller-manager-58b8ccf868-rbscn
1/1     Running   0             56s
capz-system                         capz-controller-manager-6467f986d8-dnvj4
1/1     Running   0             62s
cert-manager                        cert-manager-6888d6b69b-7b7m9
1/1     Running   0             91s
cert-manager                        cert-manager-cainjector-76f7798c9-gnp8f
1/1     Running   0             91s
cert-manager                        cert-manager-webhook-7d4b5d8484-gn5dr
1/1     Running   0             91s
gce-pd-csi-driver                   csi-gce-pd-controller-5bd587fbfb-lrx29
5/5     Running   0             5m40s
gce-pd-csi-driver                   csi-gce-pd-node-4cgd8
2/2     Running   0             4m22s
gce-pd-csi-driver                   csi-gce-pd-node-5qsfk
2/2     Running   0             4m23s
gce-pd-csi-driver                   csi-gce-pd-node-5w4bq
2/2     Running   0             4m18s
gce-pd-csi-driver                   csi-gce-pd-node-fbdbw
2/2     Running   0             4m23s
gce-pd-csi-driver                   csi-gce-pd-node-h82lx
2/2     Running   0             4m23s
gce-pd-csi-driver                   csi-gce-pd-node-jzq58
2/2     Running   0             5m39s
gce-pd-csi-driver                   csi-gce-pd-node-k6bz9
2/2     Running   0             2m39s
kube-system                         cluster-autoscaler-7f695dc48f-v5kvh
1/1     Running   0             5m40s
```

```
kube-system                         coredns-64897985d-hbkqd
1/1     Running   0             5m38s
kube-system                         coredns-64897985d-m8g5j
1/1     Running   0             5m38s
kube-system                         etcd-gcp-example-control-plane-9z77w
1/1     Running   0             5m32s
kube-system                         etcd-gcp-example-control-plane-rtj9h
1/1     Running   0             2m37s
kube-system                         etcd-gcp-example-control-plane-zbf9w
1/1     Running   0             4m17s
kube-system                         kube-apiserver-gcp-example-control-
plane-9z77w                 1/1     Running   0              5m32s
kube-system                         kube-apiserver-gcp-example-control-plane-
rtj9h                   1/1     Running   0             2m38s
kube-system                         kube-apiserver-gcp-example-control-plane-
zbf9w                   1/1     Running   0             4m17s
kube-system                         kube-controller-manager-gcp-example-
control-plane-9z77w         1/1     Running   0             5m33s
kube-system                         kube-controller-manager-gcp-example-
control-plane-rtj9h         1/1     Running   0             2m37s
kube-system                         kube-controller-manager-gcp-example-
control-plane-zbf9w         1/1     Running   0             4m17s
kube-system                         kube-proxy-bskz2
1/1     Running   0             4m18s
kube-system                         kube-proxy-gdkn5
1/1     Running   0             4m23s
kube-system                         kube-proxy-knvb9
1/1     Running   0             4m22s
kube-system                         kube-proxy-tcj7r
1/1     Running   0             4m23s
kube-system                         kube-proxy-thdpl
1/1     Running   0             5m38s
kube-system                         kube-proxy-txxmb
1/1     Running   0             4m23s
kube-system                         kube-proxy-vq6kv
1/1     Running   0             2m39s
kube-system                         kube-scheduler-gcp-example-control-
plane-9z77w                 1/1     Running   0              5m33s
kube-system                         kube-scheduler-gcp-example-control-plane-
rtj9h                   1/1     Running   0             2m37s
kube-system                         kube-scheduler-gcp-example-control-plane-
zbf9w                   1/1     Running   0             4m17s
node-feature-discovery              node-feature-discovery-master-7d5985467-
lh7dc                   1/1     Running   0              5m40s
node-feature-discovery              node-feature-discovery-worker-5qtvg
1/1     Running   0             3m40s
node-feature-discovery              node-feature-discovery-worker-66rwx
1/1     Running   0             3m40s
node-feature-discovery              node-feature-discovery-worker-7h92d
1/1     Running   0             3m35s
node-feature-discovery              node-feature-discovery-worker-b4666
1/1     Running   0             3m40s
```

```
tigera-operator                    tigera-operator-5f9bdc5c59-j9tnr
1/1     Running   0             5m38s
```

6.18.3.5.2 Next Step

GCP Install Kommander Non-air-gapped (see page 1516)

6.18.3.6 GCP Install Kommander Non-air-gapped

This section provides installation instructions for the Kommander component of DKP for a non-air-gapped environment in GCP.

6.18.3.6.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

6.18.3.6.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init > kommander.yaml
   ```

4. If required, customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include:
   - Custom Domains and Certificates
   - HTTP proxy
   - Disabling the AI Navigator application
   - External Load Balancer
   - GPU utilization, etc.

- Rook Ceph customization for Pre-provisioned environments (see page 1541)

5. *If required:* If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
...
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true
...
```

6. Expand **one** of the following sets of instructions, depending on your license and application environments:

Essential License: Install Kommander in a Non-Air-Gapped Environment

6.18.3.6.1.2 Essential License: Install Kommander

Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Enterprise License: Install Kommander in a Non-air-gapped Environment with DKP Catalog Applications

6.18.3.6.1.3 Enterprise License: Install Kommander with DKP Catalog Applications

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
...
catalog:
  repositories:
```

```
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.7.0
...
```

⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

6.18.3.6.2 Kommander Customizations

You can configure the Kommander component of DKP during the initial installation, and also post-installation using the DKP CLI. If you are not sure of what you want to customize during install, then proceed to the next step. To read about Kommander component customization options, refer to this section of the documetation: Kommander Customizations (see page 1558)

6.18.3.6.3 Next Step

Day 2 - Cluster Operations Management (see page 590)

6.18.4 GCP Management Tools

After cluster creation and configuration, you can revisit clusters to update and change variables.

- Manage GCP Node Pools (see page 1518)
- GCP Cluster Autoscaler (see page 1523)
- Delete a GCP Cluster (see page 1526)

6.18.4.1 Manage GCP Node Pools

Node pools are part of a cluster and managed as a group, and you can use a node pool to manage a group of machines using the same common properties. When Konvoy creates a new default cluster, there is one node pool for the worker nodes and all nodes in that new node pool have the same configuration. You can create additional node pools for more specialized hardware or configuration. For example, if you want to tune your memory usage on a cluster where you need maximum memory for some machines and minimal memory on other machines, you would create a new node pool with those specific resource needs.

Konvoy implements node pools using Cluster API MachineDeployments[1140].

- Create GCP Node Pools (see page 1519)
- List GCP Node Pools (see page 1520)
- Scale GCP Node Pools (see page 1521)
- Delete GCP Node Pools (see page 1523)

6.18.4.1.1 Create GCP Node Pools

Creating a node pool is useful when you need to run workloads that require machines with specific resources, such as additional memory, or specialized network or storage hardware.

6.18.4.1.1.1 Prerequisites

Before you begin, make sure you have created a GCP cluster. (see page 1503)

Prepare the environment

Follow these steps:

1. Set the environment variable to the name you assigned this cluster.

   ```
   export CLUSTER_NAME=gcp-example
   ```

2. If your workload cluster is self-managed, as described in Make the New Cluster Self-Managed (see page 1509), configure `kubectl` to use the kubeconfig for the cluster.

   ```
   export KUBECONFIG=${CLUSTER_NAME}.conf
   ```

3. Define your node pool name.

   ```
   export NODEPOOL_NAME=example
   ```

Create a GCP node pool

Availability zones (AZs) are isolated locations within data center regions from which public cloud services originate and operate. Because all the nodes in a node pool are deployed in a single Availability Zone, you may wish to create additional node pools is to ensure your cluster has nodes deployed in multiple Availability Zones.

Create a new AWS node pool with 3 replicas using this command:

Set the `--zone` flag to a zone[1141] in the same same region as your cluster.

1140 https://cluster-api.sigs.k8s.io/developer/architecture/controllers/machine-deployment.html
1141 https://cloud.google.com/compute/docs/regions-zones

```
dkp create nodepool gcp ${NODEPOOL_NAME} \
    --cluster-name=${CLUSTER_NAME} \
    --image $IMAGE_NAME \
    --zone us-west1-b \
    --replicas=3
```

```
machinedeployment.cluster.x-k8s.io/example created
⋅⋅ Creating default/example nodepool resources
gcpmachinetemplate.infrastructure.cluster.x-k8s.io/example created
kubeadmconfigtemplate.bootstrap.cluster.x-k8s.io/example created
 ✓ Creating default/example nodepool resources
```

This example uses default values for brevity. Use flags to define custom instance types, and other properties.

Advanced users can use a combination of the `--dry-run` and `--output=yaml` or or `--output-directory=<your-target-directory>/` flags to get a complete set of node pool objects to modify locally or store in version control.

Next Topic

List GCP Node Pools (see page 1520)

6.18.4.1.2 List GCP Node Pools

Use this command to list the node pools of a given cluster. This returns specific properties of each node pool so that you can see the name of the MachineDeployments.

To list all node pools for a managed cluster, run:

```
dkp get nodepool --cluster-name=${CLUSTER_NAME}
```

The expected output is similar to the following example, indicating the desired size of the node pool, the number of replicas ready in the node pool, and the Kubernetes version those nodes are running:

```
NODEPOOL                    DESIRED   READY   KUBERNETES
VERSION
example                     3         3       v1.28.7
gcp-example-2-md-0          4         4       v1.28.7
```

6.18.4.1.2.1 Next Topic

Scale GCP Node Pools (see page 1521)

6.18.4.1.3 Scale GCP Node Pools

While you can run Cluster Autoscaler[1142], you can also manually scale your node pools up or down when you need more finite control over your environment. For example, if you require 10 machines to run a process, you can manually set the scaling to run those 10 machines only. However, if also using the Cluster Autoscaler, you must stay within your minimum and maximum bounds.

6.18.4.1.3.1 Scaling Up Node Pools

To scale up a node pool in a cluster, run:

```
dkp scale nodepool ${NODEPOOL_NAME} --replicas=5 --cluster-name=${CLUSTER_NAME}
```

Your output should be similar to this example, indicating the scaling is in progress:

```
✓ Scaling node pool example to 5 replicas
```

After a few minutes, you can list the node pools to:

```
dkp get nodepool --cluster-name=${CLUSTER_NAME}
```

Your output should be similar to this example, with the number of DESIRED and READY replicas increased to 5:

```
NODEPOOL                        DESIRED             READY
KUBERNETES VERSION
example                         5                   5                   v1.28.7
gcp-example-md-0                4                   4                   v1.28.7
```

6.18.4.1.3.2 Scaling Down Node Pools

To scale down a node pool, run:

```
dkp scale nodepool ${NODEPOOL_NAME} --replicas=4 --cluster-name=${CLUSTER_NAME}
```

```
✓ Scaling node pool example to 4 replicas
```

After a few minutes, you can list the node pools using this command:

1142 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi

```
dkp get nodepool --cluster-name=${CLUSTER_NAME}
```

Your output should be similar to this example, with the number of DESIRED and READY replicas decreased to 4:

```
NODEPOOL                          DESIRED              READY
KUBERNETES VERSION
example                           4                    4                    v1.28.7
gcp-example-md-0                  4                    4                    v1.28.7
```

In a default cluster, the nodes to delete are selected at random. This behavior is controlled by CAPI's delete policy[1143]. However, when using the DKP CLI to scale down a node pool, it is also possible to specify the Kubernetes Nodes you want to delete.

To do this, set the flag `--nodes-to-delete` with a list of nodes as below. This adds an annotation `cluster.x-k8s.io/delete-machine=yes` to the matching Machine object that contains `status.NodeRef` with the node names from `--nodes-to-delete`.

```
dkp scale nodepool ${NODEPOOL_NAME} --replicas=3 --cluster-name=${CLUSTER_NAME}
```

```
✓ Scaling node pool example to 3 replicas
```

6.18.4.1.3.3 Scaling Node Pools When Using Cluster Autoscaler

If you configured the cluster autoscaler[1144] for the `demo-cluster-md-0` node pool, the value of `--replicas` must be within the minimum and maximum bounds.

For example, assuming you have these annotations:

```
kubectl annotate machinedeployment ${NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-
autoscaler-node-group-min-size=2
kubectl annotate machinedeployment ${NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-
autoscaler-node-group-max-size=6
```

Try to scale the node pool to 7 replicas with the command:

```
dkp scale nodepool ${NODEPOOL_NAME} --replicas=7 --cluster-name=${CLUSTER_NAME}
```

Which results in an error similar to:

1143 https://github.com/kubernetes-sigs/cluster-api/blob/v0.4.0/api/v1alpha4/machineset_types.go#L85-L105
1144 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi

```
 ✗ Scaling node pool example to 7 replicas
failed to scale nodepool: scaling MachineDeployment is forbidden: desired replicas 7
 is greater than the configured max size annotation cluster.x-k8s.io/cluster-api-
autoscaler-node-group-max-size: 6
```

Similarly, scaling down to a number of replicas less than the configured `min-size` also returns an error.

Next Topic

Delete GCP Node Pools (see page 1523)

6.18.4.1.4 Delete GCP Node Pools

Delete node pools in a cluster

Deleting a node pool deletes the Kubernetes nodes and the underlying infrastructure. All nodes will be drained prior to deletion and the pods running on those nodes will be rescheduled.

To delete a node pool from a managed cluster, run:

```
dkp delete nodepool ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME}
```

Here, `example` is the node pool to be deleted.

The expected output will be similar to the following example, indicating the node pool is being deleted:

```
✓ Deleting default/example nodepool resources
```

Deleting an invalid node pool results in output similar to this example:

```
dkp delete nodepool ${CLUSTER_NAME}-md-invalid --cluster-name=${CLUSTER_NAME}
```

```
MachineDeployments or MachinePools.infrastructure.cluster.x-k8s.io "no
MachineDeployments or MachinePools found for cluster gcp-example" not found
```

6.18.4.2 GCP Cluster Autoscaler

Configure autoscaler for node pools

Cluster Autoscaler[1145] provides the ability to automatically scale-up or scale-down the number of worker nodes in a cluster, based on the number of pending pods to be scheduled. Running the Cluster Autoscaler is optional.

1145 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi

Unlike Horizontal-Pod Autoscaler[1146], Cluster Autoscaler does not depend on any Metrics server and does not need Prometheus or any other metrics source.

The Cluster Autoscaler looks at the following annotations on a MachineDeployment to determine its scale-up and scale-down ranges:

```
cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size
cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size
```

The full list of command line arguments to the Cluster Autoscaler controller is on the Kubernetes public GitHub repository[1147].

For more information about how Cluster Autoscaler works, see these documents:

- What is Cluster Autoscaler[1148]
- How does scale-up work[1149]
- How does scale-down work[1150]
- CAPI Provider for Cluster Autoscaler[1151]

6.18.4.2.1 Cluster Autoscaler Prerequisites

Before you begin, you must have:

- A Bootstrap Cluster Lifecycle (see page 1501).
- A New Kubernetes Cluster (see page 1503).
- A Self-Managed Cluster (see page 1509).

6.18.4.2.2 Run Cluster Autoscaler

The Cluster Autoscaler controller runs on the workload cluster. Upon creation of the workload cluster, this controller does not have all the objects required to function correctly until after a `dkp move` is issued from the bootstrap cluster.

Run the following steps to enable Cluster Autoscaler:

1. Ensure the Cluster Autoscaler controller is up and running (no restarts and no errors in the logs)

   ```
   kubectl --kubeconfig=${CLUSTER_NAME}.conf logs deployments/cluster-autoscaler
   cluster-autoscaler -n kube-system -f
   ```

2. Enable Cluster Autoscaler by setting the min & max ranges

1146 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-fast-is-hpa-when-combined-with-ca
1147 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-are-the-parameters-to-ca
1148 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#what-is-cluster-autoscaler
1149 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-up-work
1150 https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#how-does-scale-down-work
1151 https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/cloudprovider/clusterapi

```
kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment $
{NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size=2
kubectl --kubeconfig=${CLUSTER_NAME}.conf annotate machinedeployment $
{NODEPOOL_NAME} cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size=6
```

3. The Cluster Autoscaler logs will show that the worker nodes are associated with node-groups and that pending pods are being watched.
4. To demonstrate that it is working properly, create a large deployment which will trigger pending pods (For this example we used GCP n2-standard-8 worker nodes. If you have larger worker-nodes, you should scale up the number of replicas accordingly).

```
cat <<EOF | kubectl --kubeconfig=${CLUSTER_NAME}.conf apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox-deployment
  labels:
    app: busybox
spec:
  replicas: 600
  selector:
    matchLabels:
      app: busybox
  template:
    metadata:
      labels:
        app: busybox
    spec:
      containers:
      - name: busybox
        image: busybox:latest
        command:
          - sleep
          - "3600"
        imagePullPolicy: IfNotPresent
      restartPolicy: Always
EOF
```

5. Cluster Autoscaler will scale up the number of Worker Nodes until there are no pending pods.
6. Scale down the number of replicas for `busybox-deployment`.

```
kubectl --kubeconfig ${CLUSTER_NAME}.conf scale --replicas=30 deployment/
busybox-deployment
```

7. Cluster Autoscaler starts to scale down the number of Worker Nodes after the default timeout of 10 minutes.

6.18.4.3 Delete a GCP Cluster

6.18.4.3.1 Prepare to Delete a Workload Cluster

A self-managed workload cluster cannot delete itself. If your workload cluster is self-managed, you must create a bootstrap cluster and move the cluster lifecycle services to the bootstrap cluster before deleting the workload cluster.

If you did not make your workload cluster self-managed, as described in Make the New GCP Cluster Self-Managed (see page 1509), proceed to the Delete the workload cluster (see page 0) section below.

1. Create a bootstrap cluster:

   The bootstrap cluster will host the Cluster API controllers that reconcile the cluster objects marked for deletion:

   NOTE: To avoid using the wrong `kubeconfig`, the following steps use explicit `kubeconfig` paths and contexts.

   ```
   dkp create bootstrap --with-gcp-bootstrap-credentials=true --kubeconfig
   $HOME/.kube/config
   ```

   Output:

   ```
   ✓ Creating a bootstrap cluster
   ✓ Initializing new CAPI components
   ```

2. Move the Cluster API objects from the workload to the bootstrap cluster: The cluster lifecycle services on the bootstrap cluster are ready, but the workload cluster configuration is on the workload cluster. The `move` command moves the configuration, which takes the form of Cluster API Custom Resource objects, from the workload to the bootstrap cluster. This process is also called a Pivot[1152].

   ```
   dkp move capi-resources \
       --from-kubeconfig ${CLUSTER_NAME}.conf \
       --to-kubeconfig $HOME/.kube/config
   ```

3. Use the cluster lifecycle services on the workload cluster to check the workload cluster status:

1152 https://cluster-api.sigs.k8s.io/reference/glossary.html?highlight=pivot#pivot

```
dkp describe cluster --kubeconfig $HOME/.kube/config -c ${CLUSTER_NAME}
```

Output:

```
NAME                                                                READY
SEVERITY  REASON  SINCE  MESSAGE
Cluster/gcp-example                                                 True
34s
├─ClusterInfrastructure - GCPCluster/gcp-example
├─ControlPlane - KubeadmControlPlane/gcp-example-control-plane      True
34s
│ ├─Machine/gcp-example-control-plane-6fbzn                         True
37s
│ │ └─MachineInfrastructure - GCPMachine/gcp-example-control-plane-62g6s
│ ├─Machine/gcp-example-control-plane-jf6s2                         True
37s
│ │ └─MachineInfrastructure - GCPMachine/gcp-example-control-plane-bsr2z
│ └─Machine/gcp-example-control-plane-mnbfs                         True
37s
│   └─MachineInfrastructure - GCPMachine/gcp-example-control-plane-s8xsx
└─Workers
  └─MachineDeployment/gcp-example-md-0                              True
37s
    ├─Machine/gcp-example-md-0-68b86fddb8-8glsw                     True
37s
    │ └─MachineInfrastructure - GCPMachine/gcp-example-md-0-zls8d
    ├─Machine/gcp-example-md-0-68b86fddb8-bvbm7                     True
37s
    │ └─MachineInfrastructure - GCPMachine/gcp-example-md-0-5zcvc
    ├─Machine/gcp-example-md-0-68b86fddb8-k9499                     True
37s
    │ └─MachineInfrastructure - GCPMachine/gcp-example-md-0-k8h5p
    └─Machine/gcp-example-md-0-68b86fddb8-l6vfb                     True
37s
      └─MachineInfrastructure - GCPMachine/gcp-example-md-0-9h5vn
```

After moving the cluster lifecycle services to the workload cluster, remember to use dkp with the workload cluster kubeconfig.

Persistent Volumes (PVs) are not deleted automatically by design in order to preserve your data. However, they take up storage space if not deleted. You must delete PVs manually. Information for backup of a cluster and PVs is on the page in documentation called Back up your Cluster's Applications and Persistent Volumes (see page 863) .

6.18.4.3.2 Delete the Workload Cluster

1. To delete a cluster, you would use `dkp delete cluster` and pass in the name of the cluster you are trying to delete with `--cluster-name` flag. You would use `kubectl get clusters` to get those details (`--cluster-name` and `--namespace`) of the Kubernetes cluster to delete it.
   NOTE: Do not use `dkp get clusters` since that gets you Kommander cluster details rather than Konvoy cluster details.

   ```
   kubectl get clusters
   ```

2. Use `dkp` with the bootstrap cluster to delete the workload cluster. Delete the Kubernetes cluster and wait a few minutes:

   Before deleting the cluster, dkp deletes all Services of type LoadBalancer on the cluster. To skip this step, use the flag `--delete-kubernetes-resources=false` .

   ```
   dkp delete cluster --kubeconfig $HOME/.kube/config --cluster-name $
   {CLUSTER_NAME}
   ```

   Output:

   ```
   ✓ Deleting Services with type LoadBalancer for Cluster default/gcp-example
   ✓ Deleting ClusterResourceSets for Cluster default/gcp-example
   ✓ Deleting cluster resources
   ✓ Waiting for cluster to be fully deleted
   Deleted default/gcp-example cluster
   ```

   After the workload cluster is deleted, delete the bootstrap cluster.

6.18.4.3.3 Delete the Bootstrap Cluster

1. Delete the bootstrap cluster using `dkp delete` :

```
dkp delete bootstrap --kubeconfig $HOME/.kube/config
```

Output:

```
✓ Deleting bootstrap cluster
```

6.18.4.3.4 Known Limitations

The DKP version used to create the workload cluster must match the DKP version used to delete the workload cluster.

7 Additional Kommander Configuration

The Kommander component of DKP can be configured differently depending on your environment type and other desired customizations.

7.1 Installing Kommander by Environment

This section contains instructions for installing the Kommander component in *air-gapped*, *non-air-gapped*, or *pre-provisioned* environments. It also contains instructions on how to enable *DKP catalog apps* or *DKP Insights,* if desired.

7.2 Kommander Customizations

This section contains instructions for enabling a custom configuration of a Kommander component such as custom domains or certificates, an HTTP Proxy, an external load balancer, etc.

7.3 Installing Kommander by Environment

This section provides installation instructions for the Kommander component of DKP according to your environment type.

BEFORE you install the Kommander component, answer the following three questions:

7.3.1 1. Is your Environment Air-gapped or Non-Air-gapped?

In an **air-gapped environment**, your environment is isolated from unsecured networks, like the Internet.

In a **non-air-gapped environment**, your environment has two-way access to and from the Internet.

See Air-gapped or Non-air-gapped Environment? (see page 83) for more information.

7.3.2 2. What License do you Have?

DKP Essential and **DKP Government Essential** are self-managed single-cluster Kubernetes solutions that give you a feature-rich, easy-to-deploy, and easy-to-manage entry-level cloud container platform. The DKP Essential and DKP Gov licenses give the user access to the entire Konvoy cluster environment, and to the Kommander platform application manager.

DKP Enterprise and **DKP Government Advanced** are multi-cluster solutions centered around a management cluster that manage multiple attached or managed Kubernetes clusters via a centralized management dashboard. For this license type, you will determine whether or not to use the DKP Catalog applications (see page 691).

See Licenses (see page 85) for more information.

7.3.3 3. Do you Want to Enable DKP Insights?

DKP Insights is a predictive analytics capability that **detects anomalies** that occur either in the present or future and generates an alert in the DKP UI.

See DKP Insights[1153] for more information.

7.3.4 4. Decide Whether to Install the AI Navigator Application

The AI Navigator is an AI chatbot that offers real-time, interactive communication to answer a wide range of user queries, spanning basic instructions to complex functionalities. DKP installs this application by default in non-air-gapped environments. However, you can disable it, if desired, as part of the installation of the DKP Kommander component.

Typically, we recommend that you generate a Kommander configuration file (see page 1560), so that you can customize the configuration prior to installing Kommander. You can use this method to disable AI Navigator. If you prefer to use a CLI approach to the installation, there is also a flag for disabling the application.

For security purposes, AI Navigator is **not** installed for air-gapped environments.

7.3.5 Next Step:

Identify or Change your Default StorageClass (see page 1531)

7.3.6 Identify or Change your Default StorageClass

Kommander requires a default `StorageClass`.

For the supported **cloud providers**, the Konvoy component handles the creation of a default `StorageClass`.

For **pre-provisioned** environments, the Konvoy component handles the creation of a `StorageClass` in the form of a `localvolumeprovisioner`, which is not suitable for production use. Before installing the Kommander component, you should identify and install a Kubernetes CSI[1154] compatible storage provider that is suitable for production, and then ensure it is set as the default as shown below. See Provision a Static Local Volume (see page 116) for more information.

For infrastructure driver specifics, see Default Storage Providers in DKP (see page 110).

1153 https://d2iq.atlassian.net/wiki/spaces/DINS
1154 https://kubernetes.io/docs/concepts/storage/volumes/#volume-types

7.3.6.1 Identify your StorageClass

This `StorageClass` is required to install Kommander.

1. Execute the following command to verify one is configured:

   ```
   kubectl get sc --kubeconfig ${CLUSTER_NAME}.conf
   ```

   The output should look similar to this. Note the `(default)` after the name:

   ```
   NAME               PROVISIONER       RECLAIMPOLICY   VOLUMEBINDINGMODE
   ALLOWVOLUMEEXPANSION   AGE
   ebs-sc (default)   ebs.csi.aws.com   Delete          WaitForFirstConsumer
   false                  41s
   ```

2. If the desired `StorageClass` is not set as default, add the following annotation to the `StorageClass` manifest:

   ```
   annotations:
     storageclass.kubernetes.io/is-default-class: "true"
   ```

More information on setting a `StorageClass` as default can be found at Changing the default storage class[1155] in the Kubernetes documentation.

7.3.6.2 Next Step:

Install Kommander According to your Environment (see page 1532)

7.3.7 Install Kommander According to your Environment

Select the installation type according to your environment:

- Install Kommander in an Air-gapped Environment (see page 1533)
- Install Kommander in a Non-air-gapped Environment (see page 1538)
- Install Kommander in a Pre-provisioned, Air-gapped Environment (see page 1541)
- Install Kommander in a Pre-provisioned, Non-air-gapped Environment (see page 1548)
- Install Kommander in a Small Environment (see page 1552)

1155 https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class

7.3.7.1 Install Kommander in an Air-gapped Environment

Follow these steps to install the Kommander component of DKP in an air-gapped environment (see page 83).

7.3.7.1.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry: Air-gapped Environments (see page 1536).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

7.3.7.1.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init --airgapped > kommander.yaml
   ```

4. *If required*, **customize** your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include:
   - Custom Domains and Certificates
   - HTTP proxy
   - External Load Balancer
   - GPU utilization, etc.
   - Rook Ceph customization for Pre-provisioned environments (see page 1541)

5. *If required*: If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
...
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true
...
```

6. Expand **one** of the following sets of instructions, depending on your license and application environments:

> If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

Essential License: Install Kommander in an Air-Gapped Environment

7.3.7.1.1.2 Essential License: Install Kommander in an Air-gapped Environment

Use the customized `kommander.yaml` to install DKP in an air-gapped environment:

```
dkp install kommander \
--installer-config kommander.yaml --kubeconfig=${CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.7.0.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.1.tar.gz
```

Enterprise License: Install Kommander in an Air-gapped Environment with DKP Catalog Applications

7.3.7.1.1.3 Enterprise License: Install Kommander in an Air-gapped Environment with DKP Catalog Applications

> If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add the following values to enable DKP Catalog Applications:

```
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./dkp-catalog-applications-v2.8.1.tar.gz
...
```

⚠️ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander \
--installer-config kommander.yaml --kubeconfig=${CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.1.tar.gz \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.1.tar.gz
\
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
v2.8.1.tar.gz
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

7.3.7.1.1.4 Next Step:

Verify Kommander Installation (see page 1555)

> See Install DKP Insights[1156][1561157] if you want to enable a solution that detects current and future anomalies in workload configurations or Kubernetes clusters.

7.3.7.1.2 Load the Images into Your Registry: Air-gapped Environments

Because air-gapped environments do not have direct access to the Internet, you must download, extract and load several required images to your local container registry (see page 1606), before installing DKP.

7.3.7.1.2.1 Download all Images for Air-gapped Deployments

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for prerequisites to download and then how to push the necessary images to this registry.

1. Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`) to load registry images as explained below.
2. Connectivity with clusters attaching to the management cluster is required:
   - Both management and attached clusters must be able to connect to the local registry.
   - The management cluster must be able to connect to all attached cluster's API servers.
   - The management cluster must be able to connect to any load balancers created for platform services on the management cluster.

7.3.7.1.2.2 Extract Air-gapped Images and Set Variables

Follow these steps to **extract** the air-gapped image bundles into your private registry:

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`, extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
```

1156 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/248938617/Install+DKP+Insights
1157 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/248938617/Install+DKP+Insights

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

```
cd dkp-v2.8.1
```

3. Set an environment variable with your registry address:

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

Load Images to your Private Registry - Konvoy

Before creating or upgrading a Kubernetes cluster, you need to load the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, refer to Local Registry Tools (see page 1606) page for more information.

Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME}
--to-registry-password=${REGISTRY_PASSWORD}
```

It can take a while to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

Next Step:

Install Kommander in an Air-gapped Environment (see page 1533)

7.3.7.2 Install Kommander in a Non-air-gapped Environment

Install the Kommander component of DKP in non-air-gapped environments (see page 84).

7.3.7.2.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

7.3.7.2.1.1 Create your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1558) for the deployment:

```
dkp install kommander --init > kommander.yaml
```

4. If required, customize your `kommander.yaml`.

   See Kommander Customizations (see page 1558) for customization options. Some of them include:

   - Custom Domains and Certificates
   - HTTP proxy
   - Disabling the AI Navigator application
   - External Load Balancer
   - GPU utilization, etc.
   - Rook Ceph customization for Pre-provisioned environments (see page 1541)

5. *If required:* If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
...
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true
...
```

6. Expand **one** of the following sets of instructions, depending on your license and application environments:

If your environment uses HTTP/HTTPS proxies, you must include the flags `--http-proxy`, `--https-proxy`, and `--no-proxy` and their related values in this command for it to be successful. More information is available in Configuring an HTTP/HTTPS Proxy (see page 1053).

Essential License: Install Kommander in a Non-Air-Gapped Environment

7.3.7.2.1.2 Essential License: Install Kommander

Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Enterprise License: Install Kommander in a Non-air-gapped Environment with DKP Catalog Applications

7.3.7.2.1.3 Enterprise License: Install Kommander with DKP Catalog Applications

If you want to enable DKP Catalog applications *after* installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

   ```
   ...
   catalog:
     repositories:
       - name: dkp-catalog-applications
         labels:
           kommander.d2iq.io/project-default-catalog-repository: "true"
           kommander.d2iq.io/workspace-default-catalog-repository: "true"
           kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
         gitRepositorySpec:
           url: https://github.com/mesosphere/dkp-catalog-applications
           ref:
             tag: v2.7.0
   ...
   ```

   ⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

   ```
   dkp install kommander --installer-config kommander.yaml --kubeconfig=$
   {CLUSTER_NAME}.conf
   ```

7.3.7.2.1.4

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File.
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

7.3.7.2.1.5 Next Step:

Verify Kommander Installation (see page 1555)

See Install DKP Insights[1158][1159] if you want to enable a solution that detects current and future anomalies in workload configurations or Kubernetes clusters.

7.3.7.3 Install Kommander in a Pre-provisioned, Air-gapped Environment

7.3.7.3.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Ensure you have loaded all necessary images for your configuration. See Load the Images into Your Registry: Air-gapped Environments (see page 1536).
- Note down the name of the cluster, where you want to install Kommander. If you do not know it, use `kubectl get clusters -A` to display it.

1158 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/248938617/Install+DKP+Insights
1159 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/248938617/Install+DKP+Insights

7.3.7.3.1.1 Create and Customize your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

   ```
   export CLUSTER_NAME=<your-management-cluster-name>
   ```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

   ```
   dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
   ```

3. Create a configuration file (see page 1558) for the deployment:

   ```
   dkp install kommander --init --airgapped > kommander.yaml
   ```

4. Edit the `kommander.yaml` file to include configuration overrides for the `rook-ceph-cluster`. DKP's default configuration ships Ceph with PVC based storage[1160] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner, you can install Ceph in host storage[1161] mode.

   You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

   a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

      ```
      ...
      rook-ceph-cluster:
          enabled: true
          values: |
            cephClusterSpec:
              storage:
                storageClassDeviceSets: []
                useAllDevices: true
                useAllNodes: true
                deviceFilter: "<<value>>"
      ...
      ```

   b. To assign specific storage devices on all nodes to the Ceph cluster:

      ```
      ...
      rook-ceph-cluster:
          enabled: true
      ```

1160 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
1161 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/

```
    values: |
      cephClusterSpec:
        storage:
          storageClassDeviceSets: []
          useAllNodes: true
          useAllDevices: false
          deviceFilter: "^sdb."
...
```

Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[1162]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[1163].

5. *If required*: **Customize** your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, etc.
6. *If required*: If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
...
apps:
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true
...
```

7. Expand **one** of the following sets of instructions, depending on your license and application environments:

Essential License: Install Kommander in a Pre-provisioned, Air-Gapped Environment

7.3.7.3.1.2 Essential License: Install Kommander in an Air-gapped Environment

Use the customized `kommander.yaml` to install DKP in a pre-provisioned, air-gapped environment:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.1.tar.gz \
```

1162 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
1163 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

```
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.1.tar.gz
```

Enterprise License: Install Kommander in a Pre-provisioned, Air-gapped Environment with DKP Catalog Applications

7.3.7.3.1.3 Enterprise License: Install Kommander in an Air-gapped Environment with DKP Catalog Applications

If you want to enable DKP Catalog applications **AFTER** installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add the following values to enable DKP Catalog Applications:

   ```
   ...
   catalog:
     repositories:
       - name: dkp-catalog-applications
         labels:
           kommander.d2iq.io/project-default-catalog-repository: "true"
           kommander.d2iq.io/workspace-default-catalog-repository: "true"
           kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
         path: ./dkp-catalog-applications-v2.8.1.tar.gz
   ...
   ```

   ⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

   ```
   dkp install kommander --installer-config kommander.yaml --kubeconfig=$
   {CLUSTER_NAME}.conf \
   --kommander-applications-repository ./application-repositories/kommander-
   applications-v2.8.1.tar.gz \
   --charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.1.tar.gz
   \
   --charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-
   v2.8.1.tar.gz
   ```

7.3.7.3.1.4 Next Step:

Verify Kommander Installation (see page 1555)

See Install DKP Insights[11641165] if you want to enable a solution that detects current and future anomalies in workload configurations or Kubernetes clusters.

7.3.7.3.2 Load the Images into Your Registry: Air-gapped, Pre-provisioned Environments

Because air-gapped environments do not have direct access to the Internet, you must download, extract and load several required images to your local container registry (see page 1606), before installing DKP.

7.3.7.3.2.1 Download all Images for Air-gapped Deployments

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for prerequisites to download and then how to push the necessary images to this registry.

1. Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`) to load registry images as explained below.
2. Connectivity with clusters attaching to the management cluster is required:
   - Both management and attached clusters must be able to connect to the local registry.
   - The management cluster must be able to connect to all attached cluster's API servers.
   - The management cluster must be able to connect to any load balancers created for platform services on the management cluster.

7.3.7.3.2.2 Extract Air-gapped Images and Set Variables

Follow these steps to **extract** the air-gapped image bundles into your private registry:

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

   ```
   cd dkp-v2.8.1
   ```

1164 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/248938617/Install+DKP+Insights
1165 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/248938617/Install+DKP+Insights

3. Set an environment variable with your registry address:

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

Only Pre-provisioned: Load Images for Deployments - Konvoy

For **Pre-provisioned air-gapped environments only**, you must run `konvoy-image upload artifacts` to copy the artifacts (see page 1120) onto the cluster hosts **before** you begin the Upgrade the CAPI Components (see page 1731) process later in the upgrade steps.

1. The Kubernetes image bundle will be located in `kib/artifacts/images` and you will want to verify image and artifacts.
   a. Verify the image bundles exist in `kib/artifacts/images`:

```
$ ls kib/artifacts/images/
kubernetes-images-1.28.7-d2iq.1.tar kubernetes-images-1.28.7-d2iq.1-
fips.tar
```

   b. Verify the artifacts for your OS exist in the `kib/artifacts/` directory and export the appropriate variables:

```
$ ls kib/artifacts/
1.28.7_centos_7_x86_64.tar.gz       1.28.7_redhat_8_x86_64_fips.tar.gz
containerd-1.6.28-d2iq.1-rhel-7.9-x86_64.tar.gz        containerd-1.6.28-
d2iq.1-rhel-8.6-x86_64_fips.tar.gz pip-packages.tar.gz
1.28.7_centos_7_x86_64_fips.tar.gz  1.28.7_rocky_9_x86_64.tar.gz
containerd-1.6.28-d2iq.1-rhel-7.9-x86_64_fips.tar.gz   containerd-1.6.28-
d2iq.1-rocky-9.0-x86_64.tar.gz
1.28.7_redhat_7_x86_64.tar.gz       1.28.7_ubuntu_20_x86_64.tar.gz
containerd-1.6.28-d2iq.1-rhel-8.4-x86_64.tar.gz        containerd-1.6.28-
d2iq.1-rocky-9.1-x86_64.tar.gz
1.28.7_redhat_7_x86_64_fips.tar.gz  containerd-1.6.28-d2iq.1-centos-7.9-
x86_64.tar.gz       containerd-1.6.28-d2iq.1-rhel-8.4-x86_64_fips.tar.gz
containerd-1.6.28-d2iq.1-ubuntu-20.04-x86_64.tar.gz
1.28.7_redhat_8_x86_64.tar.gz       containerd-1.6.28-d2iq.1-centos-7.9-
x86_64_fips.tar.gz  containerd-1.6.28-d2iq.1-rhel-8.6-x86_64.tar.gz
images
```

   c. Set the bundle values with the name from the private registry location:

```
export OS_PACKAGES_BUNDLE=<name_of_the_OS_package>
export CONTAINERD_BUNDLE=<name_of_the_containerd_bundle>
```

   For example, for RHEL 8.4 you would set:

```
export OS_PACKAGES_BUNDLE=1.28.7_redhat_8_x86_64.tar.gz
export CONTAINERD_BUNDLE=containerd-1.6.28-d2iq.1-rhel-8.4-x86_64.tar.gz
```

2. Upload the artifacts onto cluster hosts:

```
konvoy-image upload artifacts \
    --container-images-dir=./kib/artifacts/images/ \
    --os-packages-bundle=./kib/artifacts/${OS_PACKAGES_BUNDLE} \
    --containerd-bundle=./kib/artifacts/${CONTAINERD_BUNDLE} \
    --pip-packages-bundle=./kib/artifacts/pip-packages.tar.gz
```

Load Images to your Private Registry - Konvoy

Before creating or upgrading a Kubernetes cluster, you need to load the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, refer to Local Registry Tools (see page 1606) page for more information.

Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-password=${REGISTRY_PASSWORD}
```

Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-password=${REGISTRY_PASSWORD}
```

Next Step:

Install Kommander in a Pre-provisioned, Air-gapped Environment (see page 1541)

7.3.7.4 Install Kommander in a Pre-provisioned, Non-air-gapped Environment

Install the Kommander component of DKP in pre-provisioned (see page 84), non-air-gapped (see page 84) environments.

7.3.7.4.1 Prerequisites

- Ensure you have reviewed all Prerequisites for Install (see page 141).
- Ensure you have a default StorageClass (see page 1531).
- Note the name of the cluster where you want to install Kommander. If you do not know the cluster name, use `kubectl get clusters -A` to display and find it.

You **MUST** modify the Kommander installer configuration file (`kommander.yaml`) before installing the Kommander component of DKP in a pre-provisioned environment.

7.3.7.4.1.1 Create and Customize your Kommander Installer Configuration File

1. Set the environment variable for your cluster:

```
export CLUSTER_NAME=<your-management-cluster-name>
```

2. Copy the `kubeconfig` file of your Management cluster to your local directory:

```
dkp get kubeconfig -c ${CLUSTER_NAME} > ${CLUSTER_NAME}.conf
```

3. Create a configuration file (see page 1558) for the deployment:

```
dkp install kommander --init > kommander.yaml
```

4. Edit the installer file to include configuration overrides for the `rook-ceph-cluster`. DKP's default configuration ships Ceph with PVC based storage[1166] which requires your CSI provider to support PVC with type `volumeMode: Block`. As this is not possible with the default local static provisioner (see page 110), you can install Ceph in host storage[1167] mode.

   You can choose whether Ceph's object storage daemon (osd) pods should consume all or just some of the devices on your nodes. Include **one** of the following Overrides:

   a. To automatically assign all raw storage devices on all nodes to the Ceph cluster:

```
...
rook-ceph-cluster:
    enabled: true
    values: |
      cephClusterSpec:
        storage:
          storageClassDeviceSets: []
          useAllDevices: true
          useAllNodes: true
          deviceFilter: "<<value>>"
...
```

   b. To assign specific storage devices on all nodes to the Ceph cluster:

```
...
rook-ceph-cluster:
    enabled: true
    values: |
      cephClusterSpec:
        storage:
          storageClassDeviceSets: []
          useAllNodes: true
          useAllDevices: false
```

1166 https://rook.io/docs/rook/v1.10/CRDs/Cluster/pvc-cluster/
1167 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/

```
          deviceFilter: "^sdb."
...
```

Note: If you want to assign specific devices to specific nodes using the `deviceFilter` option, refer to Specific Nodes and Devices[1168]. For general information on the `deviceFilter` value, refer to Storage Selection Settings[1169].

5. *If required:* Customize your `kommander.yaml`.
   See Kommander Customizations (see page 1558) for customization options. Some of them include: Custom Domains and Certificates, HTTP proxy, External Load Balancer, GPU utilization, etc.
6. *If required:* If your cluster uses a custom **AWS VPC** and requires an internal load-balancer, set the `traefik` annotation to create an internal-facing ELB:

```
...
apps:
...
  traefik:
    enabled: true
    values: |
      service:
        annotations:
          service.beta.kubernetes.io/aws-load-balancer-internal: "true
...
```

7. Expand **one** of the following sets of instructions, depending on your license and application environments:

Essential License: Install Kommander in a Pre-provisioned, Non-Air-Gapped Environment

7.3.7.4.1.2 Essential License: Install Kommander

Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Enterprise License: Install Kommander in a Pre-provisioned, Non-air-gapped Environment with DKP Catalog Applications

1168 https://rook.io/docs/rook/v1.10/CRDs/Cluster/host-cluster/#specific-nodes-and-devices
1169 https://rook.io/docs/rook/v1.10/CRDs/Cluster/ceph-cluster-crd/#storage-selection-settings

7.3.7.4.1.3 Enterprise License: Install Kommander with DKP Catalog Applications

If you want to enable DKP Catalog applications after installing DKP, see Enable DKP Catalog Applications after Installing DKP (see page 1595).

1. In the same `kommander.yaml` of the previous section, add these values for `dkp-catalog-applications`:

```
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.7.0
...
```

⚠ If you only want to enable catalog applications to an **existing** configuration, add these values to an **existing** installer configuration file to maintain your Management cluster's settings.

2. Use the customized `kommander.yaml` to install DKP:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).

- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

7.3.7.4.1.4 Next Step:

Verify Kommander Installation (see page 1555)

See Install DKP Insights[11701171] if you want to enable a solution that detects current and future anomalies in workload configurations or Kubernetes clusters.

7.3.7.5 Install Kommander in a Small Environment

You can install the Kommander component of DKP on a small, single-cluster environment with smaller memory, storage, and CPU requirements for testing and demo purposes. This topic describes methods for installing Kommander in these environments.

Enterprise considerations: D2iQ recommends performing testing and demo tasks in a single-cluster environment. The Enterprise license is designed for multi-cluster environments and fleet management, which require a minimum number of resources (see page 122). Applying an Enterprise license key to the previous installation adds modifications to your environment that can exhaust a small environment's resources.

7.3.7.5.1 Prerequisites

Ensure you have done the following:

- You have acquired a DKP license.
- You have installed the Konvoy component (see page 146).
- You have reviewed the prerequisite section (see page 141) pertaining to your air-gapped, or networked environment.

1170 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/248938617/Install+DKP+Insights
1171 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/248938617/Install+DKP+Insights

7.3.7.5.2 Minimal Kommander installation

The YAML file that is used to install a minimal configuration of Kommander contains the bare minimum setup that allows you to deploy applications, and access the DKP UI. It does **NOT** include applications for cost monitoring, logging, alerting, object storage, etc.

In this YAML file you can find the lines that correspond to all platform applications which would be included in a normal Kommander setup. Applications that have `enabled` set to `false` are not taken into account during installation. If you want to test an additional application, you can enable it individually to be installed by setting `enabled` to `true` on the corresponding line in the YAML file.

For example, if you want to enable the logging stack, set `enabled` to `true` for `grafana-logging`, `grafana-loki` `logging-operator`, `rook-ceph` and `rook-ceph-cluster`. Note that depending on the size of your cluster, enabling several platform applications could exhaust your cluster's resources.

Some applications depend on other applications to work properly. Refer to the dependencies documentation (see page 669) to find out which other applications you need to enable to test the target application.

1. Initialize your Kommander installation and name it `kommander_minimal.yaml`:

```
dkp install kommander --init --kubeconfig=${CLUSTER_NAME}.conf -oyaml >
kommander_minimal.yaml
```

2. Edit your `kommander_minimal.yaml` file to match the following example:

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  dex:
    enabled: true
  dex-k8s-authenticator:
    enabled: true
  dkp-insights-management:
    enabled: false
  gatekeeper:
    enabled: true
  gitea:
    enabled: true
  grafana-logging:
    enabled: false
  grafana-loki:
    enabled: false
```

```
  kommander:
    enabled: true
  kommander-ui:
    enabled: true
  kube-prometheus-stack:
    enabled: false
  kubernetes-dashboard:
    enabled: false
  kubefed:
    enabled: true
  kubetunnel:
    enabled: false
  logging-operator:
    enabled: false
  prometheus-adapter:
    enabled: false
  reloader:
    enabled: true
  rook-ceph:
    enabled: false
  rook-ceph-cluster:
    enabled: false
  traefik:
    enabled: true
  traefik-forward-auth-mgmt:
    enabled: true
  velero:
    enabled: false
ageEncryptionSecretName: sops-age
clusterHostname: ""
```

3. Install Kommander on your cluster with the following command:

```
dkp install kommander --installer-config ./kommander_minimal.yaml --
kubeconfig=${CLUSTER_NAME}.conf
```

In the previous command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to **install Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 106).

TIP: Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

If the Kommander installation fails, or you wish to reconfigure applications, you can rerun the install command, and you can view the progress by increasing the log verbosity by adding the flag `-v 2`.

7.3.7.5.3 Next Step:

Verify Kommander Installation (see page 1555)

7.3.8 Verify Kommander Installation

After you build the Konvoy cluster and you install Kommander, verify your installation. After the CLI successfully installs the components, it waits for all applications to be ready by default.

> **NOTE**: If the Kommander installation fails or you wish to reconfigure applications, you can rerun the install command to retry the installation.

If you prefer the CLI to not wait for all applications to become ready, you can set the `--wait=false` flag.

If you choose not to wait via the DKP CLI, you can check the status of the installation using the following command:

```
kubectl -n kommander wait --for condition=Ready helmreleases --all --timeout 15m
```

This will wait for each of the helm charts to reach their `Ready` condition, eventually resulting in an output resembling below:

```
helmrelease.helm.toolkit.fluxcd.io/centralized-grafana condition met
helmrelease.helm.toolkit.fluxcd.io/dex condition met
helmrelease.helm.toolkit.fluxcd.io/dex-k8s-authenticator condition met
helmrelease.helm.toolkit.fluxcd.io/fluent-bit condition met
helmrelease.helm.toolkit.fluxcd.io/gitea condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-logging condition met
helmrelease.helm.toolkit.fluxcd.io/grafana-loki condition met
helmrelease.helm.toolkit.fluxcd.io/karma condition met
helmrelease.helm.toolkit.fluxcd.io/kommander condition met
helmrelease.helm.toolkit.fluxcd.io/kommander-appmanagement condition met
helmrelease.helm.toolkit.fluxcd.io/kube-prometheus-stack condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost condition met
helmrelease.helm.toolkit.fluxcd.io/kubecost-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/kubefed condition met
helmrelease.helm.toolkit.fluxcd.io/kubernetes-dashboard condition met
helmrelease.helm.toolkit.fluxcd.io/kubetunnel condition met
helmrelease.helm.toolkit.fluxcd.io/logging-operator condition met
```

```
helmrelease.helm.toolkit.fluxcd.io/logging-operator-logging condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-adapter condition met
helmrelease.helm.toolkit.fluxcd.io/prometheus-thanos-traefik condition met
helmrelease.helm.toolkit.fluxcd.io/reloader condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph condition met
helmrelease.helm.toolkit.fluxcd.io/rook-ceph-cluster condition met
helmrelease.helm.toolkit.fluxcd.io/thanos condition met
helmrelease.helm.toolkit.fluxcd.io/traefik condition met
helmrelease.helm.toolkit.fluxcd.io/traefik-forward-auth-mgmt condition met
helmrelease.helm.toolkit.fluxcd.io/velero condition met
```

7.3.8.1 Failed HelmReleases

If an application fails to deploy, check the status of a `HelmRelease` with:

```
kubectl -n kommander get helmrelease <HELMRELEASE_NAME>
```

If you find any `HelmReleases` in a "broken" release state, such as "exhausted" or "another rollback/release in progress", trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": true}]'
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"op":
"replace", "path": "/spec/suspend", "value": false}]'
```

7.3.8.2 Next Step:

Log in to the UI with Kommander (see page 1556)

7.3.9 Log in to the UI with Kommander

By default, you can log in to the UI in Kommander with the credentials given using this command:

```
dkp open dashboard --kubeconfig=${CLUSTER_NAME}.conf
```

You can also retrieve your credentials at any time using the following command:

```
kubectl -n kommander get secret dkp-credentials -o go-template='Username:
{{.data.username|base64decode}}{{ "\n"}}Password: {{.data.password|base64decode}}
{{ "\n"}}'
```

You can also retrieve the URL used for accessing the UI using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='https://{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}/dkp/kommander/
dashboard{{ "\n"}}'
```

You should only use these static credentials to access the UI for configuring an external identity provider (see page 609). Treat them as back up credentials rather than use them for normal access.

You rotate the password using the following command:

```
dkp experimental rotate dashboard-password
```

The output displays the new password:

```
Password: kqZ31lMBSCLcBjUKVwLJMQL2PxalipIzZw5Pjyw09wDqjWV3dz2wPSSBYi09JGJp
```

You can perform the following operations on Identity Providers (see page 609):

- Create an Identity Provider
- Temporarily Disable an Identity Provider
- Create Groups

7.3.9.1 Dashboard UI Functions

After installing Konvoy component and building a cluster as well as successfully installing Kommander and logging into the UI, you are now ready to customize configurations using the Day 2 Cluster Operations Management (see page 590) section of the documentation. The majority of this customization such as attaching clusters and deploying applications will take place in the dashboard or UI of DKP. The Day 2 section allows you to manage cluster operations and their application workloads to optimize your organization's productivity.

See Install DKP Insights[1172] if you want to enable a solution that detects current and future anomalies in workload configurations or Kubernetes clusters.

7.3.9.2 Next Step

Day 2 - Cluster Operations Management (see page 590)

1172 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/248938617/Install+DKP+Insights

7.4 Kommander Customizations

You can configure the Kommander component of DKP during the initial installation, and also post-installation using the DKP CLI.

7.4.1 Prerequisites

- Review the Management cluster application requirements (see page 122) to ensure your cluster has enough resources.
- Ensure you have a default `StorageClass`, as shown in Identify or Change your Default StorageClass (see page 1531).

7.4.2 Initialize a *Kommander Installer Configuration File*

To begin configuring Kommander, run the following command to initialize a default configuration file:

- For a non-air-gapped environment, run the following command:

```
dkp install kommander --init > kommander.yaml
```

- For an air-gapped environment, run the following command:

```
dkp install kommander --init --airgapped > kommander.yaml
```

7.4.3 Configure Applications

After you have a default configuration file, you can then configure each `app` either inline **or** by referencing another YAML file. The configuration values for each `app` correspond to the Helm Chart values for the application.

After the initial deployment of Kommander, you can find the application Helm Charts by checking the `spec.chart.spec.sourceRef` field of the associated `HelmRelease`:

```
kubectl get helmreleases <application> -o yaml -n kommander
```

7.4.3.1 Inline configuration (using values)

In this example, you configure the `centralized-grafana` application with resource limits by defining the Helm Chart values in the Kommander configuration file.

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  centralized-grafana:
    values: |
      grafana:
        resources:
          limits:
            cpu: 150m
            memory: 100Mi
          requests:
            cpu: 100m
            memory: 50Mi
...
```

7.4.3.2 Reference another YAML file (using valuesFrom)

Alternatively, you could create another YAML file containing the configuration for `centralized-grafana` and reference that using `valuesFrom`. Point to this file by using either a relative path (from the configuration file location) or by using an absolute path.

```
cat > centralized-grafana.yaml <<EOF
grafana:
  resources:
    limits:
      cpu: 150m
      memory: 100Mi
    requests:
      cpu: 100m
      memory: 50Mi
EOF
```

```
apiVersion: config.kommander.mesosphere.io/v1alpha1
kind: Installation
apps:
  centralized-grafana:
    valuesFrom: centralized-grafana.yaml
...
```

7.4.4 Minimal Kommander Installation

You can install Kommander with a bare minimum of applications on a small environment with smaller memory, storage, and CPU requirements for testing and demo purposes. Refer to the Install DKP on a Small Environment (see page 1552) documentation for more information.

7.4.5 Install with Configuration File

In the following command, the `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you set the context to **install Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 106).

Add the `--installer-config` flag to the `kommander install` command to use a custom configuration file. To reconfigure applications, you can also run this command after the initial installation.

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

TIP: Sometimes, applications require a longer period of time to deploy, which causes the installation to time out. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.

7.4.6 Verify Installation

After building the Konvoy cluster and installing Kommander, you can verify your Kommander installation (see page 1555), then you can Log in to the Kommander UI. (see page 1556)

7.4.7 DKP Kommander Configuration Reference

7.4.7.1 Configuration Parameters

This page contains the configuration parameters for the Kommander component of DKP.

For additional information about configuring the Kommander component of DKP during initial installation, see Kommander Customizations (see page 1558).

Parameter	Description	Default Value

apps
(AppConfig list (see page 1563))
	List of platform applications that will be installed on the management cluster.	<pre>apps:
		  ai-navigator-app:
		    enabled: true
		  dex:
		    enabled: true
		  dex-k8s-authenticator:
		    enabled: true
		  dkp-insights-management:
		    enabled: true
		  gatekeeper:
		    enabled: true
		  gitea:
		    enabled: true
		  grafana-logging:
		    enabled: true
		  grafana-loki:
		    enabled: true
		  kommander:
		    enabled: true
		  kube-prometheus-stack:
		    enabled: true
		    values: <shortened for brevity>
		  kubefed:
		    enabled: true
		  kubernetes-dashboard:
		    enabled: true
		  kubetunnel:
		    enabled: true
		  logging-operator:
		    enabled: true
		  prometheus-adapter:
		    enabled: true
		  reloader:
		    enabled: true
		   rook-ceph:
		    enabled: true
		   rook-ceph-cluster:
		    enabled: true
		  traefik:
		    enabled: true
		  traefik-forward-auth-mgmt:
		    enabled: true
		  velero:
		    enabled: true</pre>

Parameter	Description	Default Value
ageEncryptionSecretName	Defines the name of the secret to store the Age encryption in.	`sops-age`
clusterHostName	Allows users to provide a hostname that is used for accessing the cluster's ingresses.	
ingressCertificate (see page 1564)	Allows users to provide a custom certificate that's used for TLS in the cluster's ingresses.	
acme	Enable automatic ingress certificate management via ACME.	
appManagementImageTag	Specifies image tag of the AppManagement container.	
appManagementImageRepository	Specifies the image repository of AppManagement container	
appManagementKubetoolsImageRepository	Specifies the image repository of AppManagement Kubetools container	
kommanderChartsVersion	Specifies DKP Kommander Helm chart version.	
airgapped (see page 1565)	Specifies parameters for an airgapped environment.	

Parameter	Description	Default Value
catalog Enterprise Gov Advanced	Specifies parameters for installing default catalog repositories. For additional information, see Enable DKP Catalog Applications after Installing DKP (see page 1595).	

7.4.7.2 AppConfig

Parameter	Description	Default Value
enabled	Denotes whether the specific app should be deployed or not. The `ai-navigator-app` entry defaults to `true` unless you are installing in an air-gapped environment. Set the value to `false` if you do not want to install the AI Navigator (see page 1976) application.	false

Parameter	Description	Default Value
valuesFrom	File path containing the values that are passed onto the application's `HelmRelease`. This a Helm values files for all applications at the moment. The path in this field must either be a relative file location which is then interpreted to be relative to the location of the configuration file or an absolute path. Only one of `valuesFrom` or `values` may be set; both cannot be set.	
values	Contains the values that are passed to the the application's `HelmRelease`. Only one of `valuesFrom` or `values` may be set; both cannot be set.	

7.4.7.3 IngressCertificate

Parameter	Description	Default Value
certificate	The path to a certificate PEM file.	
private_key	The path to the certificate’s private key (PEM).	

Parameter	Description	Default Value
ca	The path to the certificate's CA bundle; a PEM file containing root and intermediate certificates.	

7.4.7.4 Airgapped

Parameter	Description	Default Value
enabled	Specifies if installation happens in an air-gapped environment.	
helmMirrorImageTag	Specifies an image tag of Helm-mirror container.	
helmMirrorImageRepository	Specifies image repository of Helm-mirror container.	

7.4.7.4.1 Next Step:

Configure HTTP Proxy (see page 1589)

7.4.8 Custom Domains and Certificates for Management and Essential Clusters

Configure a custom domain and certificate during the installation of the Kommander component. For Management (see page 80) or Essential (see page 81) clusters only.

There are two configuration methods:

Configuration Method	*WHILE* installing the Kommander component	*AFTER* installing the Kommander component
Supported cluster types	Only Essential or Management clusters	All cluster types
Documentation	Remain in this section	Go to Custom Domains and Certificates Configuration for All Cluster Types (see page 888)

DKP supports configuring a custom **domain** name for accessing the UI and other platform services, as well as setting up manual or automatic **certificate renewal** or rotation. This section provides instructions and examples on how to configure the DKP installation to add a customized domain and certificate on your **Essential cluster** (see page 81) or your **Management cluster** (see page 80).

Section contents:

7.4.8.1 Why to set up a Custom Domain or Certificate?

7.4.8.1.1 Reasons for Using a Custom DNS Domain

DKP supports the customization of domains to allow you to use your own domain or hostname for your services. For example, you can set up your DKP UI or any of your clusters to be accessible with your custom domain name instead of the domain provided by default.

- To set up a custom domain (without a custom certificate), refer to Configure a Custom Domain without a Custom Certificate (see page 1576).

7.4.8.1.2 Reasons for Using a Custom Certificate

DKP's default CA identity supports the encryption of data exchange and traffic (between your client and your environment's server). To configure an additional security layer that validates your environment's server authenticity, DKP supports configuring a custom certificate issued by a trusted Certificate Authority either directly in a Secret or managed automatically using the ACME protocol (for example, Let's Encrypt).

Changing the default certificate for any of your clusters can be helpful. For example, you can adapt it to classify your DKP UI or any other type of service as trusted (when accessing a service via a browser).

To set up a custom domain and certificate, refer to the following pages respectively:

- Configure a custom domain and certificate as part of the cluster's installation process (see page 1569). This is only possible for your Management/Essential cluster (see page 79).
- Update your cluster's current domain and certificate configuration as part of your cluster's Day 2 operations (see page 891). You can do this for any cluster type (see page 79) in your environment.

Using Let's Encrypt or other public ACME certificate authorities **does not work in air-gapped scenarios**, as these services require connection to the Internet for their setup. For air-gapped environments, you can either use self-signed certificates issued by the cluster (the default configuration), or a certificate created manually using a trusted Certificate Authority.

7.4.8.1.3 Next Topic:

Certificate Issuer and KommanderCluster Concepts (see page 1567)

7.4.8.2 Certificate Issuer and KommanderCluster Concepts

If you set up an advanced configuration of your custom domain (see page 1570), ensure you understand the following concepts.

7.4.8.2.1 `KommanderCluster` Object

The `KommanderCluster` resource is an object that contains key information for all types of clusters (see page 79) that are part of your environment, such as:

- Cluster access and endpoint information
- Cluster attachment information
- Cluster status and configuration information

7.4.8.2.2 Issuer Objects: `Issuer`, `ClusterIssuer` or `certificateSecret`?

If you use a certificate issued and managed automatically by `cert-manager`, you need an `Issuer` or `ClusterIssuer` that you reference in your `KommanderCluster` resource. The referenced object must contain the information of your certificate provider.

If you want to use a manually-created certificate, you need a `certificateSecret` that you reference in your `KommanderCluster` resource.

7.4.8.2.3 Location of the `KommanderCluster` and Issuer Objects: Management, Managed or Attached cluster?

In the **Management** (see page 80) or **Essential** (see page 81) cluster, both the `KommanderCluster` and *issuer* objects are stored on the same cluster. The *issuer* can be referenced as an `Issuer`, `ClusterIssuer` or `certificateSecret`.

In **Managed** (see page 80) and **Attached** (see page 80) clusters, the `KommanderCluster` object is stored on the Management cluster. The `Issuer`, `ClusterIssuer` or `certificateSecret` is stored on the Managed or Attached cluster.

7.4.8.2.4 HTTP or DNS solver?

When configuring a certificate for your DKP cluster, you can set up an *HTTP solver* or a *DNS solver*. The HTTP protocol exposes your cluster to the public Internet, whereas DNS keeps your traffic hidden. If you use HTTP, your cluster must be publically accessible (via the ingress or load balancer). If you use DNS, this is not a requirement. See Advanced Configuration: ClusterIssuer (see page 1574) for HTTP and DNS configuration options.

> If you are enabling a proxied access (see page 846) for a network-restricted cluster (see page 82), this configuration is restricted to DNS.

7.4.8.2.5 Related topics:

Why to set up a Custom Domain or Certificate? (see page 1566)

Configure the Kommander Installation with a Custom Domain and Certificate (see page 1569)

Advanced Configuration: ClusterIssuer (see page 1574)

Certificate Authority (CA) Specifics (see page 1568)

7.4.8.3 Certificate Authority (CA) Specifics

The following table defines some values you require to set up a custom certificate according to your Certificate Authority (CA).

Certificate Authority	**Prerequisites**	***Kommander Installer* values**
Let's Encrypt	None	Generated automatically by Kommander when `acme` is enabled
ZeroSSL	An access and a secret key provided by ZeroSSL	`acme.server: https://acme.zerossl.com/v2/DV90`
SSL.com	An access and a secret key provided by SSL	`acme.server: https://acme.ssl.com/sslcom-dv-rsa`

Use these values to Configure your Custom Domain and Certificate (see page 1569).

7.4.8.3.1 Next Topic:

Configure your Custom Domain and Certificate (see page 1569)

7.4.8.4 Configure the Kommander Installation with a Custom Domain and Certificate

This page contains instructions on how to set up custom certificates for your Management (see page 80) or Essential (see page 81) cluster during the installation of DKP.

There are two configuration methods:

Configuration Method	*WHILE* installing the Kommander component	*AFTER* installing the Kommander component
Supported cluster types	Only Essential or Management clusters	All cluster types
Documentation	Remain in this page	Go to Configure Custom Domains or Custom Certificates post Kommander Installation (see page 891)

7.4.8.4.1 Configuration Options

- Choose an ACME-supported Certificate Authority, if you want the `cert-manager` to automatically handle **certificate renewal** and **rotation**.
- Refer to Certificate Authority (CA) Specifics (see page 1568) for more information on values that are specific to your Certificate Authority or CA.

I want to use an automatically-generated certificate with ACME and require basic configuration*

7.4.8.4.1.1 I want to use an **automatically-generated** certificate with **ACME** and require **basic** configuration*

When you enable ACME, by default DKP generates an ACME-supported certificate with an HTTP01 solver. The `cert-manager` automatically issues a trusted certificate for the configured custom domain, and takes care of renewing the certificate before expiration.

1. Open the *Kommander Installer Configuration File* or `<kommander.yaml>` file:

a. If you do not have the `<kommander.yaml>` file, initialize the configuration file (see page 1558), so you can edit it in the following steps. **WARNING**: Initialize this file only ONE time, otherwise you will overwrite previous customizations.
b. If you have initialized the configuration file already, open the `<kommander.yaml>` with the editor of your choice.

2. In that file, configure the custom domain for your cluster:

```
[...]
clusterHostname: <mycluster.example.com>
[...]
```

3. Enable ACME by adding `acme` value, the issuer's server and your e-mail. If you don't provide a server, DKP sets up **Let's Encrypt** as your certificate provider:

```
acme:
  email: <your_email>
  server: <your_server>
[...]
```

4. Use the configuration file to install Kommander (see page 0).

*basic configuration: ACME server without EAB (External Account Bindings) and HTTP solver

I want to use an automatically-generated certificate with ACME and require advanced configuration (e.g. EAB, DNS solver, etc.)

7.4.8.4.1.2 I want to use an **automatically-generated** certificate with **ACME** and require **advanced** configuration

If you require additional configuration options like DNS solver, EAB, among others, create a `ClusterIssuer` with the required configurations before you run the installation of Kommander. The `cert-manager` automatically issues a trusted certificate for the configured custom domain, and takes care of renewing the certificate before expiration.

To read more about the `ClusterIssuer`, other objects, and where to store them, refer to Advanced Configuration: ClusterIssuer (see page 1574) and Certificate Issuer and KommanderCluster Concepts (see page 1567).

1. Create a `ClusterIssuer` and store it in the target cluster. It **must** be called `kommander-acme-issuer`:

a. If you require an **HTTP** solver, adapt the following example with the properties required for your certificate and execute the command:

```
cat <<EOF | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: kommander-acme-issuer # This part is important
spec:
  acme:
    email: <your_email>
    server: <https://acme.server.example>
    skipTLSVerify: true
    privateKeySecretRef:
      name: kommander-acme-issuer-account # Set this to <name>-account
    solvers:
    - http01:
        ingress:
          ingressTemplate:
            metadata:
              annotations:
                kubernetes.io/ingress.class: kommander-traefik
                "traefik.ingress.kubernetes.io/router.priority":
"2147483647"
EOF
```

⚠ The values `kommander-acme-issuer`, `kommander-acme-issuer-account` and `"traefik.ingress.kubernetes.io/router.priority": "2147483647"` are not placeholders and MUST be filled out exactly as in the example.

ℹ In **on-premise** environments, replace the annotation in the previous example with `traefik.ingress.kubernetes.io/router.tls: "true"`.

b. If you require a **DNS** solver, adapt the following example with the properties required for your certificate and execute the command:

```
cat <<EOF | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: kommander-acme-issuer # This part is important
spec:
  acme:
    email: <your_email>
    server: <https://acme.server.example>
    privateKeySecretRef:
      name: kommander-acme-issuer-account # Set this to <name>-account
    solvers:
      - dns01:
          route53:
```

```
            region: us-east-1
            role: arn:aws:iam::YYYYYYYYYYYY:role/dns-manager
EOF
```

⚠ The values `kommander-acme-issuer`, `kommander-acme-issuer-account` are not placeholders and MUST be filled out exactly as in the example.

2. Optional: If you require External Account Bindings to link your ACME account to an external database, refer to https://cert-manager.io/docs/configuration/acme/#external-account-bindings.
3. Optional: Create a DNS record, by setting up the external-dns service (see page 1579). This way, the `external-dns` will take care of pointing the DNS record to the ingress of the cluster automatically. ℹ You can also create a DNS record manually, that maps your domain name or IP address to the cluster ingress. If you choose to create a DNS record manually, finish installing the Kommander component, and then manually create a DNS record that points to the load balancer address.
4. Open the **Kommander Installer Configuration File** or `kommander.yaml` file:
   a. If you do not have the `kommander.yaml` file, initialize the configuration file (see page 1558), so you can edit it in the following steps. **WARNING**: Initialize this file only ONCE, otherwise you will overwrite previous customizations.
   b. If you have initialized the configuration file already, open the `kommander.yaml` with the editor of your choice.
5. In that file, configure the cluster to use your custom domain:

```
[...]
clusterHostname: <mycluster.example.com>
[...]
```

6. Enable ACME by configuring the issuer's server and your e-mail:

```
[...]
acme:
  email: <your_email>
  server: <your_server>
[...]
```

7. Use the configuration file to install Kommander (see page 0).

I have a manually-generated certificate

7.4.8.4.1.3 I have a **manually-generated** certificate

D2iQ supports the use of a manually-created certificate. In this case, there is no certificate controller that handles the renewal and update of your certificate automatically, so you will have to take care of these tasks manually.

Prerequisites:

- Obtain the PEM files of your certificate and store them in the target cluster's namespace:
  - Certificate
  - certificate's private key
  - CA bundle (containing the root and intermediate certificates)

Configure the manually-generated certificate

1. Open the **Kommander Installer Configuration File** or `<kommander.yaml>` file:
   a. If you do not have the `<kommander.yaml>` file, initialize the configuration file (see page 1558), so you can edit it in the following steps. **WARNING**: Initialize this file only ONCE, otherwise you will overwrite previous customizations.
   b. If you have initialized the configuration file already, open the `<kommander.yaml>` with the editor of your choice.
2. In the *Kommander Installer Configuration file*, provide your custom domain and the paths to the PEM files of your certificate:

```
[...]
clusterHostname: <mycluster.example.com>
ingressCertificate:
  certificate: <certs/cert.pem>
  private_key: <certs/key.pem>
  ca: <certs/ca.pem>
[...]
```

3. Use the configuration file to install Kommander (see page 0).

Certificates issued by another Issuer

You can also configure a certificate issued by another Certificate Authority. In this case, the CA will determine which information to include in the configuration.

- Refer to https://cert-manager.io/docs/configuration/ for configuration examples.
- The `ClusterIssuer`'s name **MUST BE** `kommander-acme-issuer`.

7.4.8.4.1.4 Next Step:

Verify and Troubleshoot the Domain and Certificate Customization (see page 1577)

7.4.8.4.1.5 Related Topics:

- Advanced Configuration: ClusterIssuer (see page 1574)
- Configure a Custom Domain without a Custom Certificate (see page 1576)

7.4.8.4.2 Advanced Configuration: ClusterIssuer

Ensure you review Certificate Issuer and KommanderCluster Concepts (see page 1567) to understand the basic concepts for certificate configuration.

When you enable ACME (see page 1569), by default DKP generates an ACME-supported certificate with an HTTP01 solver that is provided by Let's Encrypt.

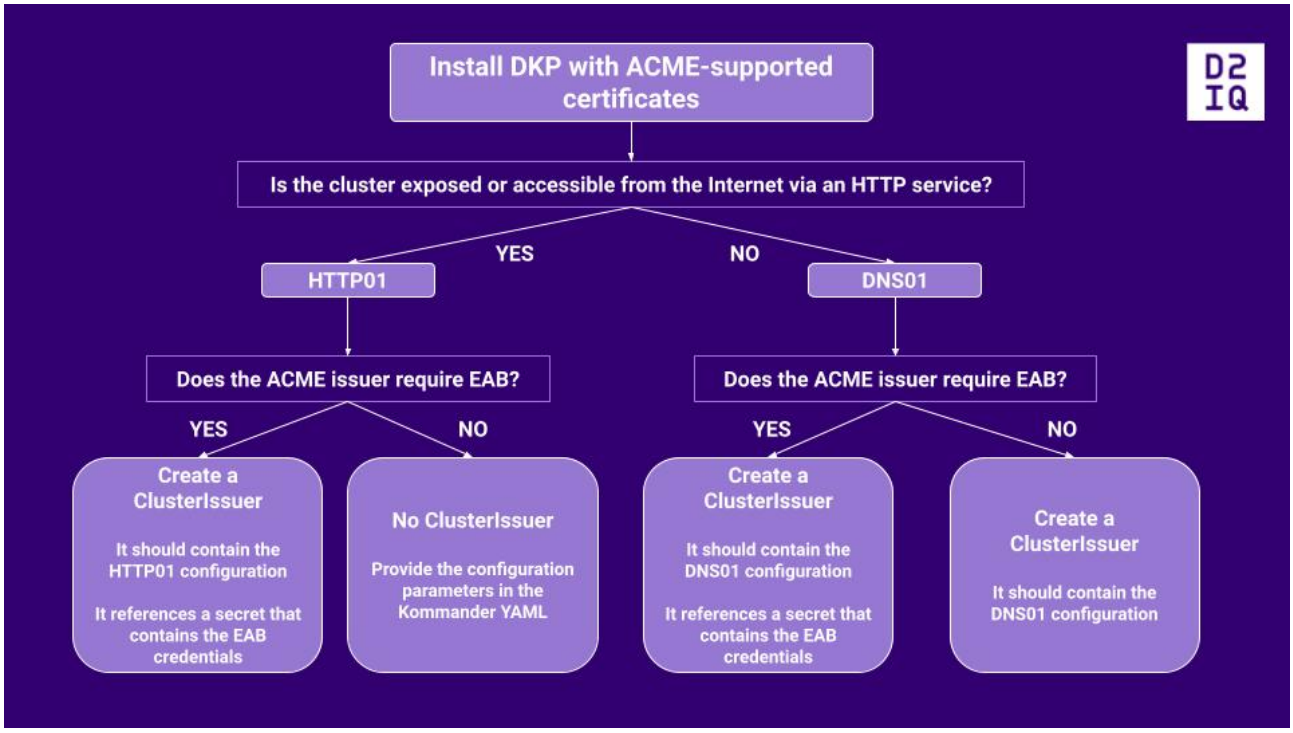
You can also set up an advanced configuration for a Custom Domain and Certificate (see page 1570). In these cases, the custom configuration cannot be done completely via the `installer config` file, but must be specified further in a `ClusterIssuer`.

Whether it is sufficient to establish the configuration of your custom certificate in the `installer config` file only, or you require a `ClusterIssuer` to define further configuration options depends on the degree of customization.

If you require a `ClusterIssuer`, you MUST create it before you run the Kommander installation.

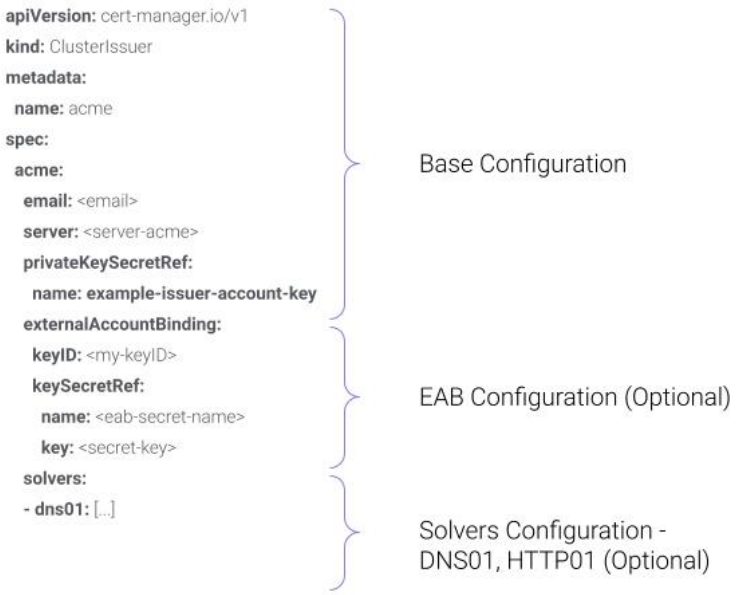
7.4.8.4.2.1 When do You Need a ClusterIssuer?

The configuration of the `ClusterIssuer` resource depends on your DKP landscape:



7.4.8.4.2.2 How do You Configure a ClusterIssuer?

The following image describes the configurable fields of a `ClusterIssuer`:



For more information on the available options, refer to the ACME section in the cert-manager documentation[1173].

7.4.8.4.2.3 Examples:

Refer to Configure the Kommander Installation with a Custom Domain and Certificate (see page 1569) for configuration steps and examples.

> ⚠ If you need to make changes in the configuration of your domain or certificate *after* you have installed DKP, or if you want to set up a custom domain and certificate for Attached or Managed clusters (see page 79), modify the `ingress` in the `KommanderCluster` object as shown in the Custom domains and certificates configuration (see page 891) section.

7.4.8.4.2.4 Related topics:

Why to set up a Custom Domain or Certificate? (see page 1566)

Configure the Kommander Installation with a Custom Domain and Certificate (see page 1569)

Certificate Issuer and KommanderCluster Concepts (see page 1567)

7.4.8.4.3 Configure a Custom Domain without a Custom Certificate

To configure Kommander to use a custom domain, the domain name must be provided in an installation config file. If you want to set up a custom domain and certificate, refer to Configure your Custom Domain and Certificate (see page 1569).

1. Open the *Kommander Installer Configuration File* or `<kommander.yaml>` file:
   a. If you do not have the `<kommander.yaml>` file, initialize the configuration file (see page 1558), so you can edit it in the following steps. **WARNING**: Initialize this file only ONCE, otherwise you will overwrite previous customizations.
   b. If you have initialized the configuration file already, open the `<kommander.yaml>` with the editor of your choice.
2. In that file, configure the custom domain for your cluster by adding this line:

```
[...]
clusterHostname: <mycluster.example.com>
[...]
```

3. This configuration can be used when installing or reconfiguring Kommander by passing it to the `dkp install kommander` command:

1173 https://cert-manager.io/docs/configuration/acme/

```
dkp install kommander --installer-config <kommander.yaml> --kubeconfig=$
{CLUSTER_NAME}.conf
```

Note: To ensure Kommander is installed on the right cluster, use the `--kubeconfig=cluster_name.conf` flag as an alternative to KUBECONFIG.

4. After the command completes, obtain the cluster ingress IP address or hostname using the following command:

```
kubectl -n kommander get svc kommander-traefik -o go-template='{{with
index .status.loadBalancer.ingress 0}}{{or .hostname .ip}}{{end}}{{ "\n"}}'
```

If required, create a DNS record (for example, by using external-dns (see page 1576)) for your custom hostname that resolves to the cluster ingress load balancer hostname or IP address. If the previous command returns a hostname, you should create a CNAME DNS entry that resolves to that hostname. If the cluster ingress is an IP address, create a DNS A record.

The domain must be resolvable from the client (your browser) and from the cluster. If you set up an `external-dns` service, it will take care of pointing the DNS record to the ingress of the cluster automatically. If you are manually creating a DNS record, you have to install Kommander first to obtain the load balancer address required for the DNS record. The Configure a Custom Certificate (see page 1569) page contains more details and examples on how and when to set up the DNS record.

7.4.8.4.3.1 Related topics:

Why to set up a Custom Domain or Certificate? (see page 1566)

Configure your Custom Domain and Certificate (see page 1569)

Advanced Configuration: Important Concepts (see page 1567)

Advanced Configuration: ClusterIssuer (see page 1574)

7.4.8.5 Verify and Troubleshoot the Domain and Certificate Customization

If you want to ensure the customization for a domain and certificate is completed, or if you want to obtain more information on the status of the customization, display the status information for the `KommanderCluster`. On the Management cluster (see page 80):

1. Inspect the modified `KommanderCluster` object:

```
kubectl describe kommandercluster -n <workspace_name> <cluster_name>
```

2. If the ingress is still being provisioned, the output looks similar to this:

```
[...]
Conditions:
  Last Transition Time:  2022-06-24T07:48:31Z
  Message:               Ingress service object was not found in the cluster
  Reason:                IngressServiceNotFound
  Status:                False
  Type:                  IngressAddressReady
[...]
```

If the provisioning has been completed, the output looks similar to this:

```
[...]
Conditions:
  Last Transition Time:  2022-06-28T13:43:33Z
  Message:               Ingress service address has been provisioned
  Reason:                IngressServiceAddressFound
  Status:                True
  Type:                  IngressAddressReady
  Last Transition Time:  2022-06-28T13:42:24Z
  Message:               Certificate is up to date and has not expired
  Reason:                Ready
  Status:                True
  Type:                  IngressCertificateReady
[...]
```

The same command also prints the actual customized values for the `KommanderCluster.Status.Ingress`. Here is an example:

```
[...]
   ingress:
    address: 172.20.255.180
    caBundle: LS0tLS1CRUdJTiBD...<output has been shortened>...DQVRFLS0tLS0K
[...]
```

7.4.8.5.1 Related topics:

Why to set up a Custom Domain or Certificate? (see page 1566)

Configure the Kommander Installation with a Custom Domain and Certificate (see page 1569)

Certificate Issuer and KommanderCluster Concepts (see page 1567)

Advanced Configuration: ClusterIssuer (see page 1574)

7.4.9 DNS Record Creation with External DNS

This section describes how to use `external-dns` to maintain your DNS records

When you set up a custom domain (see page 888) for your cluster, you require a DNS record that maps the configured domain or IP address to the cluster's ingress. You can either create one manually, or set up the `external-dns` service to manage your DNS record automatically.

If you choose to use `external-dns` to maintain your DNS records, the `external-dns` will take care of pointing the DNS record to the ingress of the cluster automatically.

Select one of the following options to configure the `external-dns` service for Management (see page 80)/ Essential (see page 81) clusters, or for Managed (see page 80)/Attached (see page 80) clusters:

- Configure External DNS with the CLI: Management or Essential Cluster (see page 1579)
- Configure External DNS with the UI: All Clusters (see page 1581)
- Verify your External DNS Configuration (see page 1583)

If you choose to create a DNS record manually, finish installing the Kommander component, and then manually create a DNS record that points to the load balancer address.

7.4.9.1 Configure External DNS with the CLI: Management or Essential Cluster

This page contains information on how to configure an `external-dns` service to manage DNS records automatically in your Management (see page 80) or Essential (see page 81) Cluster.

7.4.9.1.1 Prerequisite

- Ensure you have configured a DNS zone with your cloud provider.

7.4.9.1.2 Configure External DNS and Customize Traefik

The configuration varies depending on your cloud provider.

1. Open the `kommander.yaml` file:
   a. If you have not installed the Kommander component yet, initialize the configuration file (see page 1558), so you can edit it in the following steps.
   ⚠️ **WARNING**: Initialize this file only ONCE, otherwise you will overwrite previous customizations.
   b. If you have installed the Kommander component already, open the existing `kommander.yaml` with the editor of your choice.
2. Adjust the `app` section of your `kommander.yaml` file to include these values:

AWS Example

Replace the placeholders `<...>` with your environment's information.

The following example shows how to configure `external-dns` to manage DNS records in AWS Route 53 automatically.

```
apps:
  external-dns:
    enabled: true
    values: |
      aws:
        credentials:
          secretKey: <secret-key>
          accessKey: <access-key>
        region: <provider-region>
        preferCNAME: true
      policy: upsert-only
      txtPrefix: local-
      domainFilters:
        - <example.com>
```

Azure Example

Replace the placeholders `<...>` with your environment's information.

```
apps:
  external-dns:
    enabled: true
    values: |
      azure:
        cloud: AzurePublicCloud
        resourceGroup: <resource-group>
        tenantId: <tenant-id>
        subscriptionId: <your-subscription-id>
        aadClientId: <client-id>
        aadClientSecret: <client-secret>
      domainFilters:
        - <example.com>
      txtPrefix: txt-
      policy: sync
      provider: azure
```

3. In the same `app` section, adjust the `traefik` section to include the following:

```
traefik:
    enabled: true
    values: |
      service:
```

```
        annotations:
          external-dns.alpha.kubernetes.io/hostname: <mycluster.example.com>
```

4. Use the configuration file to install or update the Kommander component:

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Refer to the external-dns documentation[1174] for more information, as well as further instructions on how to configure `external-dns` to use other DNS providers like Google Cloud DNS, CloudFlare, or on-site providers.

7.4.9.1.2.1 Next Step:

Verify your External DNS Configuration (see page 1583)

7.4.9.2 Configure External DNS with the UI: All Clusters

Enterprise Gov Advanced

This page contains information on how to configure an `external-dns` service to manage DNS records automatically and applies to all cluster types (see page 79).

7.4.9.2.1 Prerequisite

- Ensure you have configured a DNS zone with your cloud provider.

7.4.9.2.2 Configure External DNS Using the UI

The configuration varies depending on your cloud provider.

1. Select the target workspace from the top navigation bar.
   It must be the workspace that contains the cluster, for which you want to configure External DNS. In the case of the Management cluster, it would be the Management cluster workspace.
2. Select **Applications** from the sidebar menu.
3. Search for the **External DNS** application.
4. On the application card, select the *three-dot menu* > **Enable**.
5. On the **Enable Workspace Platform Application** page, select **Configuration** from the sidebar menu.
6. Copy and paste the following contents into the code editor and replace the placeholders `<...>` with your environment's information.

1174 https://artifacthub.io/packages/helm/bitnami/external-dns

Here is an example configuration:

AWS Example

Replace the placeholders `<...>` with your environment's information.

The following example shows how to configure `external-dns` to manage DNS records in AWS Route 53 automatically.

```
aws:
  credentials:
    secretKey: <secret-key>
    accessKey: <access-key>
  region: <provider-region>
  preferCNAME: true
policy: upsert-only
txtPrefix: local-
domainFilters:
  - <example.com>
```

Azure Example

Replace the placeholders `<...>` with your environment's information.

```
azure:
  cloud: AzurePublicCloud
  resourceGroup: <resource-group>
  tenantId: <tenant-id>
  subscriptionId: <your-subscription-id>
  aadClientId: <client-id>
  aadClientSecret: <client-secret>
domainFilters:
  - <example.com>
txtPrefix: txt-
policy: sync
provider: azure
```

Refer to external-dns documentation[1175] for more configuration options.

1175 https://artifacthub.io/packages/helm/bitnami/external-dns

7.4.9.2.3 Customize the Traefik Deployment Using the UI

DKP deploys Treafik to all clusters by default.

1. Select the target workspace from the top navigation bar.
   It must be the workspace that contains the cluster, for which you want to configure External DNS. In the case of the Management cluster, it would be the Management cluster workspace.
2. Select **Applications** from the sidebar menu.
3. Search for the *Traefik* application.
4. On the application card, select the *three-dot menu* > **Edit**.
5. Select the **Configuration** lateral tab to add a customization.
6. Copy and paste the following configuration into the code editor:
   Use the **Cluster Application Configuration Override** code editor to apply a configuration per cluster.

```
service:
  annotations:
    external-dns.alpha.kubernetes.io/hostname: <mycluster.example.com>
```

Ensure you set up a domain per cluster, for example: `<mycluster1.example.com>`, `<mycluster2.example.com>` and `<mycluster3.example.com>`.

7.4.9.2.3.1 Next Step:

Verify your External DNS Configuration (see page 1583)

7.4.9.3 Verify your External DNS Configuration

This page contains commands to verify that the `external-dns` service is functioning correctly.

If the `external-dns` service is not working properly, these commands also provide aids to find the cause or identify the issue.

7.4.9.3.1 Verify the Deployment

Verify that the deployment was triggered.

1. Set the environment variable to the **Management/Essential cluster** by exporting the `kubeconfig` file in your terminal window or using the `--kubeconfig=${CLUSTER_NAME}.conf` as explained in Provide Context for Commands with a kubeconfig File (see page 106).
2. Verify that the `external-dns` deployment is present:
   Replace `<target_WORKSPACE_NAMESPACE>` in the namespace `-n` flag with the target cluster's workspace namespace.

   ```
   kubectl get appdeployments.apps.kommander.d2iq.io -n
   <target_WORKSPACE_NAMESPACE> external-dns
   ```

   The output should look like this:

   ```
   NAME           APP                            AGE
   external-dns   external-dns-<app_version>   36s
   ```

   The CLI has triggered the deployment of the application. However, this does not mean that the application has been installed completely and successfully.

Verify that the deployment was successful.

1. Set the environment variable to the **target cluster** (where you enabled `external-dns`) by exporting the `kubeconfig` file in your terminal window or using the `--kubeconfig=${CLUSTER_NAME}.conf` as explained in Provide Context for Commands with a kubeconfig File (see page 106).
2. Verify that the `external-dns` deployment is ready:
   Replace `<target_WORKSPACE_NAMESPACE>` in the namespace `-n` flag with the target cluster's workspace namespace.

   ```
   kubectl get deployments.apps -n <target_WORKSPACE_NAMESPACE> external-dns
   ```

   The deployment should display a ready state:

   ```
   NAME           READY   UP-TO-DATE   AVAILABLE   AGE
   external-dns   1/1     1            1           42s
   ```

   The CLI has deployed the application completely and successfully.

7.4.9.3.2 Examine the Cluster's Ingress

1. Set the environment variable to the **target cluster** (where you enabled `external-dns`) by exporting the `kubeconfig` file in your terminal window or using the `--kubeconfig=$ {CLUSTER_NAME}.conf` as explained in Provide Context for Commands with a kubeconfig File (see page 106).
2. Verify that the cluster's ingress contains the correct hostname annotation:
   Replace `<target_WORKSPACE_NAMESPACE>` in the namespace `-n` flag with the target cluster's workspace namespace.

   ```
   kubectl get services -n <target_WORKSPACE_NAMESPACE> kommander-traefik -o yaml
   ```

   The output looks like this:
   Ensure that the service object contains the `external-dns.alpha.kubernetes.io/ hostname: <mycluster.example.com>` annotation.

   ```
   apiVersion: v1
   kind: Service
   metadata:
     annotations:
       meta.helm.sh/release-name: kommander-traefik
       meta.helm.sh/release-namespace: kommander
       external-dns.alpha.kubernetes.io/hostname: <mycluster.example.com>
     creationTimestamp: "2023-06-21T04:52:49Z"
     finalizers:
   [...]
   ```

   The `external-dns` service has been linked to the cluster correctly.

7.4.9.3.3 Verify the DNS Record

> It can take a few minutes for the `external-dns` service to create a DNS record. The delay depends on your cloud provider.

Verify that the `external-dns` service has created a DNS record.

1. Set the environment variable to the **target cluster** (where you enabled `external-dns`) by exporting the `kubeconfig` file in your terminal window or using the `--kubeconfig=$`

`{CLUSTER_NAME}.conf` as explained in Provide Context for Commands with a kubeconfig File (see page 106).

2. Access and execute the required image:

```
kubectl run -it --image=nicolaka/netshoot --rm test-dns -- /bin/bash
```

3. Use the image to check your domain and see the record:
   Replace `<mycluster.example.com>` with the domain you assigned to your target cluster.

```
nslookup <mycluster.example.com>
```

   The output should look like this:

```
Server:     192.168.178.1
Address:    192.168.178.1#53

Non-authoritative answer:
Name:   <mycluster.example.com>
Address: 134.568.789.12
```

   The `external-dns` service is working and the DNS provider recognizes the record created by the service. If the command displays an error, the configuration is failing on the end of the DNS provider.

Trobleshooting

If your deployment has not succeeded and the previous steps have not helped you identify the issue, you can also check the logs for the `external-dns` deployment:

1. Set the environment variable to the **target cluster** (where you enabled `external-dns`) by exporting the `kubeconfig` file in your terminal window or using the `--kubeconfig=${CLUSTER_NAME}.conf` as explained in Provide Context for Commands with a kubeconfig File (see page 106).
2. Verify the `external-dns` logs:

   Replace `<target_WORKSPACE_NAMESPACE>` in the namespace `-n` flag with the target cluster's workspace namespace.

```
kubectl logs -n kommander deployment/external-dns
```

   The output displays the pod's logs for the `external-dns` deployment. Here is an example:

```
...
```

```
time="2023-07-04T06:56:35Z" level=info msg="Instantiating new Kubernetes
client"
time="2023-07-04T06:56:35Z" level=info msg="Using inCluster-config based on
serviceaccount-token"
time="2023-07-04T06:56:35Z" level=info msg="Created Kubernetes client https://
10.96.0.1:443"
time="2023-07-04T06:56:35Z" level=error msg="records retrieval failed: failed
to list hosted zones:
...
```

7.4.10 External Load Balancer

7.4.10.1 Load Balancing for External Traffic in DKP

DKP includes a load balancing solution for the supported cloud infrastructure providers (see page 67) and for pre-provisioned environments. For more information, see Load Balancing for external traffic (see page 994) in DKP.

If you want to use a **non-DKP load balancer** (for example, as an alternative to MetalLB in pre-provisioned environments), DKP supports setting up an **external load balancer**.

When enabled, the external load balancer routes incoming traffic requests to a single point of entry in your cluster. Users and services can then access the **DKP UI** through an established IP or DNS address.

In DKP environments, the external load balancer must be configured without TLS termination.

7.4.10.2 Configure Kommander to use an External Load Balancer

To configure an external load balancer, configure a custom hostname (static IP or dynamic DNS address) and specify the target `nodePorts` for your cluster.

1. Open the **Kommander Installer Configuration File** or `kommander.yaml` file:
   a. If you do not have the `kommander.yaml` file, initialize the configuration file (see page 1558), so you can edit it in the following steps. **WARNING**: Initialize this file only ONCE, otherwise you will overwrite previous customizations.
   b. If you have initialized the configuration file already, open the `kommander.yaml` with the editor of your choice.
2. In that file, add the following line for the IP address or DNS name:
   ⚠ ACME does not support the automatic creation of a certificate if you select an IP address for your `clusterHostname`.

```
[...]
clusterHostname: <mycluster.example.com OR IP_address>
[...]
```

3. Optional: If you require a custom certificate for your `clusterHostname`, see Configure the Kommander Installation with a Custom Domain and Certificate (see page 1569).
4. In the same **Kommander Installer Configuration File**, configure Kommander to use the `NodePort` service by adding a custom configuration under `traefik`:

   ⚠ You can specify the `nodePort` entry points for the load balancer. Ensure the port is within the Kubernetes default (30 000 - 32 768). If not specified, Kommander assigns a port dynamically.

   ```
   traefik:
     enabled: true
     values: |-
       ports:
         web:
           nodePort: 32080 #if not specified, will be assigned dynamically
         websecure:
           nodePort: 32443 #if not specified, will be assigned dynamically
       service:
         type: NodePort
   ```

5. Use the configuration file to install Kommander (see page 1560).

7.4.10.3 Configure the External Load Balancer to Target the Specified Ports

The `traefik` service of the Kommander component now actively listens on the pod IPs, and is accessible through the specified ports on every node.

Configure the load balancer targets to include every worker node address (DNS name or IP address) and node port combination by following this format:

```
<node1>:<nodePort_web> # for example, my.node1.internal:32080
<node2>:<nodePort_web>
<node3>:<nodePort_web>
[...]
<node1>:<nodePort_websecure> # for example, my.node1.internal:32443
<node2>:<nodePort_websecure>
<node3>:<nodePort_websecure>
[...]
```

The exact configuration depends on your load balancer provider.

7.4.11 Installing Kommander with an HTTP Proxy

Configure HTTP proxy for the Kommander clusters

Kommander supports environments that connect through an HTTP/HTTPS proxy, when access to the Internet is restricted. Use the information in this section to configure the Kommander component of DKP correctly.

In these environments, you must configure Kommander to use the HTTP/HTTPS proxy. In turn, Kommander configures all platform services to use the HTTP/HTTPS proxy.

Kommander follows a common convention for using an HTTP proxy server. The convention is based on three environment variables, and is supported by many, though not all, applications.

- `HTTP_PROXY` : the HTTP proxy server address
- `HTTPS_PROXY` : the HTTPS proxy server address
- `NO_PROXY` : a list of IPs and domain names that are not subject to proxy settings

7.4.11.1 Prerequisites

In the examples below:

1. The `curl` command-line tool is available on the host.
2. The proxy server address is `http://proxy.company.com:3128` .
3. The HTTP and HTTPS proxy server addresses use the `http` scheme.
4. The proxy server can reach `www.google.com` using HTTP or HTTPS.

Verify the cluster nodes can access the Internet through the proxy server. On each cluster node, run:

```
curl --proxy http://proxy.company.com:3128 --head http://www.google.com
curl --proxy http://proxy.company.com:3128 --head https://www.google.com
```

If the proxy is working for HTTP and HTTPS, respectively, the `curl` command returns a `200 OK` HTTP response.

7.4.11.2 Enable Gatekeeper

Gatekeeper acts as a Kubernetes mutating webhook[1176]. You can use this to mutate the Pod resources with `HTTP_PROXY`, `HTTPS_PROXY` and `NO_PROXY` environment variables.

1. Create (if necessary) or update the Kommander installation configuration file. If one does not already exist, then create it using the following commands:

   ```
   dkp install kommander --init > kommander.yaml
   ```

2. Append this `apps` section to the `kommander.yaml` file with the following values to enable Gatekeeper and configure it to add HTTP proxy settings to the pods.

   NOTE: Only pods created after applying this setting will be mutated. Also, this will only affect pods in the namespace with the `"gatekeeper.d2iq.com/mutate=pod-proxy"` label.

   ```
   apps:
     gatekeeper:
       values: |
         disableMutation: false
         mutations:
           enablePodProxy: true
           podProxySettings:
             noProxy:
   "127.0.0.1,192.168.0.0/16,10.0.0.0/16,10.96.0.0/12,169.254.169.254,169.254.0.0/
   24,localhost,kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.de
   fault.svc.cluster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.c
   luster.local,.svc.cluster.local.,kubecost-prometheus-server.kommander,logging-
   operator-logging-fluentd.kommander.svc.cluster.local,elb.amazonaws.com"
             httpProxy: "http://proxy.company.com:3128"
             httpsProxy: "http://proxy.company.com:3128"
           excludeNamespacesFromProxy: []
           namespaceSelectorForProxy:
             "gatekeeper.d2iq.com/mutate": "pod-proxy"
   ```

3. Create the `kommander` and `kommander-flux` namespaces, or the namespace where Kommander will be installed. Label the namespaces to activate the Gatekeeper mutation on them:

   ```
   kubectl create namespace kommander
   kubectl label namespace kommander gatekeeper.d2iq.com/mutate=pod-proxy

   kubectl create namespace kommander-flux
   kubectl label namespace kommander-flux gatekeeper.d2iq.com/mutate=pod-proxy
   ```

1176 https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/#mutatingadmissionwebhook

7.4.11.3 Create Gatekeeper ConfigMap in the Kommander Namespace

To configure Gatekeeper so that these environment variables are mutated in the pods, create the following `gatekeeper-overrides` ConfigMap in the `kommander` Workspace you created in a previous step:

```
export NAMESPACE=kommander
```

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: gatekeeper-overrides
  namespace: ${NAMESPACE}
data:
  values.yaml: |
    ---
    # enable mutations
    disableMutation: false
    mutations:
      enablePodProxy: true
      podProxySettings:
        noProxy:
"127.0.0.1,192.168.0.0/16,10.0.0.0/16,10.96.0.0/12,169.254.169.254,169.254.0.0/24,loc
alhost,kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cl
uster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.cluster.local,.svc.
cluster.local.,kubecost-prometheus-server.kommander,logging-operator-logging-
fluentd.kommander.svc.cluster.local,elb.amazonaws.com"
        httpProxy: "http://proxy.company.com:3128"
        httpsProxy: "http://proxy.company.com:3128"
      excludeNamespacesFromProxy: []
      namespaceSelectorForProxy:
        "gatekeeper.d2iq.com/mutate": "pod-proxy"
EOF
```

Set the `httpProxy` and `httpsProxy` environment variables to the address of the HTTP and HTTPS proxy servers, respectively. Set the `noProxy` environment variable to the addresses that should be accessed directly, not through the proxy.

Performing this step before installing Kommander allows the Flux components to respect the proxy configuration in this ConfigMap.

7.4.11.4 HTTP Proxy Configuration Considerations

To ensure that core components work correctly, always add these addresses to the `noProxy`:

- Loopback addresses (`127.0.0.1` and `localhost`)

- Kubernetes API Server addresses
- Kubernetes Pod IPs (for example, `192.168.0.0/16`). This comes from two places:
  - Calico pod CIDR - Defaults to `192.168.0.0/16`
  - The `podSubnet` is configured in CAPI objects and needs to match above Calico's - Defaults to `192.168.0.0/16` (same as above)
- Kubernetes Service addresses (for example, `10.96.0.0/12`, `kubernetes`, `kubernetes.default`, `kubernetes.default.svc`, `kubernetes.default.svc.cluster`, `kubernetes.default.svc.cluster.local`, `.svc`, `.svc.cluster`, `.svc.cluster.local`, `.svc.cluster.local.`)
- Auto-IP addresses `169.254.169.254,169.254.0.0/24`

In addition to the values above, the following settings are needed when installing on AWS:

- The default VPC CIDR range of `10.0.0.0/16`
- `kube-apiserver` internal/external ELB address

- The `NO_PROXY` variable contains the Kubernetes Services CIDR. This example uses the default CIDR, `10.96.0.0/12`. If your cluster's CIDR is different, update the value in the `NO_PROXY` field.
- Based on the order in which the Gatekeeper Deployment is Ready (in relation to other Deployments), not all the core services are guaranteed to be mutated with the proxy environment variables. Only the user deployed workloads are guaranteed to be mutated with the proxy environment variables. If you need a core service to be mutated with your proxy environment variables, you can restart the AppDeployment for that core service.

7.4.11.5 Install Kommander

Kommander installs with the DKP CLI. Install Kommander using the configuration files and ConfigMap from previous steps:

NOTE: To ensure Kommander is installed on the workload cluster, use the `--kubeconfig=cluster_name.conf` flag:

```
dkp install kommander --installer-config kommander.yaml
```

7.4.11.6 Configure Workspace or Project

Configure the Workspace or Project in which you want to use the proxy. To have Gatekeeper mutate the manifests, create the `Workspace` (or `Project`) with the following label:

```
labels:
  gatekeeper.d2iq.com/mutate: "pod-proxy"
```

This can be done when creating the Workspace (or Project) from the UI OR by running the following command from the CLI after creating the namespace:

```
kubectl label namespace <NAMESPACE> "gatekeeper.d2iq.com/mutate=pod-proxy"
```

7.4.11.7 Configure HTTP Proxy in Attached Clusters

To ensure that Gatekeeper is deployed before everything else in the attached clusters that you want to configure with proxy configuration, you must manually create the exact Namespace of the Workspace in which the cluster is going to be attached, *before* attaching the cluster:

Execute the following command in the attached cluster before attaching it to the host cluster:

```
kubectl create namespace <NAMESPACE>
```

Then, to configure the pods in this namespace to use proxy configuration, you must label the Workspace with `gatekeeper.d2iq.com/mutate=pod-proxy` when creating it so that Gatekeeper deploys a `validatingwebhook` to mutate the pods with proxy configuration.

```
kubectl label namespace <NAMESPACE> "gatekeeper.d2iq.com/mutate=pod-proxy"
```

7.4.11.8 Create Gatekeeper ConfigMap in the Workspace Namespace

To configure Gatekeeper so that these environment variables are mutated in the pods, create the following `gatekeeper-overrides` ConfigMap in the Workspace Namespace:

```
export NAMESPACE=<NAMESPACE>
```

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: gatekeeper-overrides
  namespace: ${NAMESPACE}
data:
  values.yaml: |
    ---
    # enable mutations
    disableMutation: false
    mutations:
      enablePodProxy: true
      podProxySettings:
        noProxy:
"127.0.0.1,192.168.0.0/16,10.0.0.0/16,10.96.0.0/12,169.254.169.254,169.254.0.0/24,loc
alhost,kubernetes,kubernetes.default,kubernetes.default.svc,kubernetes.default.svc.cl
uster,kubernetes.default.svc.cluster.local,.svc,.svc.cluster,.svc.cluster.local,.svc.
cluster.local.,kubecost-prometheus-server.kommander,logging-operator-logging-
fluentd.kommander.svc.cluster.local,elb.amazonaws.com"
        httpProxy: "http://proxy.company.com:3128"
        httpsProxy: "http://proxy.company.com:3128"
      excludeNamespacesFromProxy: []
      namespaceSelectorForProxy:
        "gatekeeper.d2iq.com/mutate": "pod-proxy"
EOF
```

Set the `httpProxy` and `httpsProxy` environment variables to the address of the HTTP and HTTPS proxy servers, respectively. Set the `noProxy` environment variable to the addresses that should be accessed directly, not through the proxy. The list of the recommended settings is in the section *HTTP Proxy Configuration Considerations* above.

7.4.11.9 Configure Your Applications

In a default installation with `gatekeeper` enabled, you can have proxy environment variables applied to all your pods automatically by adding the following label to your namespace:

```
"gatekeeper.d2iq.com/mutate": "pod-proxy"
```

No further manual changes are required.

7.4.11.10 Manually Configure Your Application

If Gatekeeper is not installed, and you need to use an HTTP proxy, you must manually configure your applications.

Some applications follow the convention of `HTTP_PROXY`, `HTTPS_PROXY`, and `NO_PROXY` environment variables.

In this example, the environment variables are set for a container in a Pod:

See Define Environment Variables for a Container[1177] for more details.

7.4.11.10.1 Next Steps:

Now select your environment, and finish your Kommander Installation using one of the following:

Install Kommander in an Air-gapped Environment (see page 1533)

Install Kommander in a Non-air-gapped Environment (see page 1538)

Install Kommander in a Small Environment (see page 1552)

7.4.12 Enable DKP Catalog Applications after Installing DKP

Enterprise Gov Advanced

DKP supports configuring DKP Catalog applications for environments with an Enterprise license. There are two possible ways of enabling the Enterprise catalog:

Enable an Enterprise catalog ***during*** the installation of DKP.	See Install Kommander in an Air-gapped Environment (see page 1533), or Install Kommander in a Non-air-gapped Environment (see page 1538) depending on your environment.
Enable an Enterprise catalog ***after*** installing DKP.	Follow the instructions on this page.

7.4.12.1 Configure a Default Enterprise Catalog after Installing DKP

1. Locate the **Kommander Installer Configuration file** you are using for your current deployment. This file is stored locally on your computer.
   The default file name is `kommander.yaml`, but your file's name could be different.
2. Open that file and go into edit mode.
3. Enable the default catalog repository by adding the following values to your existing and already configured **Kommander Installer Configuration file**:

```
...
```

1177 https://kubernetes.io/docs/tasks/inject-data-application/define-environment-variable-container/#define-an-environment-variable-for-a-container

```
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      gitRepositorySpec:
        url: https://github.com/mesosphere/dkp-catalog-applications
        ref:
          tag: v2.7.0
...
```

4. Reconfigure DKP by reinstalling the Kommander component. In the following example, replace `kommander.yaml` with the name of your **Kommander Installer Configuration file.**
⚠️ Ensure you are using the correct name for the **Kommander Installer Configuration file** to maintain your clusters settings. Installing a different `kommander.yaml` than the one your environment is using overwrites all of your previous configurations and customizations. ⚠️

```
dkp install kommander --installer-config kommander.yaml --kubeconfig=$
{CLUSTER_NAME}.conf
```

Tips and recommendations

- The `--kubeconfig=${CLUSTER_NAME}.conf` flag ensures that you **install Kommander on the correct cluster**. For alternatives, see Provide Context for Commands with a kubeconfig File (see page 106).
- Applications can take longer to deploy, and time out the installation. Add the `--wait-timeout <time to wait>` flag and specify a period of time (for example, `1h`) to allocate more time to the deployment of applications.
- If the Kommander installation fails, or you wish to reconfigure applications, rerun the `install` command to retry.

7.4.12.2 DKP Catalog Application labels

The following section describes each label:

Label	Description
`kommander.d2iq.io/project-default-catalog-repository`	Indicates this acts as a Catalog Repository in all projects
`kommander.d2iq.io/workspace-default-catalog-repository`	Indicates this acts as a Catalog Repository in all workspaces
`kommander.d2iq.io/gitapps-gitrepository-type`	Indicates this Catalog Repository (and all its Applications) are certified to run on DKP

7.4.12.3 Next Step:

Custom Domains and Certificates for Management and Essential Clusters (see page 1565)

8 Additional Konvoy Configurations

When installing DKP for a project, line-of-business, or enterprise, the first step is to determine the infrastructure on which you want to deploy.

The infrastructure you select then determines the specific requirements for a successful installation. Day 1 - Basic Install (see page 146) section gave you a recommended install. For further customization or advanced questions, this Day 1 section of the documentation will provide answers.

If you have decided to uninstall DKP, refer to the same infrastructure documentation you have selected in this Day 1 section for specific steps to take down your cluster.

The different Infrastructures and components for further configuration are listed below:

8.1 Section Contents

8.2 FIPS 140-2 Compliance

Understand FIPS-140 Operating Mode and Requirements

Developed by a working group of government, industry operators, and vendors, the Federal Information Processing Standard (FIPS), FIPS-140 defines security requirements for cryptographic modules. FIPS defines what cryptographic cyphers can be used. Kubernetes uses encryption by default between various components and FIPS support ensures that the ciphers used for those communications meet those standards. The standard provides for a wide spectrum of data sensitivity, transaction values, and a diversity of application environment security situations. The standard specifies four security levels for each of eleven requirement areas. Each successive level offers increased security.

NIST introduced FIPS 140-2 validation, by accredited third party laboratories, as a formal, rigorous process to protect sensitive digitally-stored information not under Federal security classifications.

8.2.1 FIPS Support in DKP

DKP supports provisioning a FIPS-enabled Kubernetes control plane. Core Kubernetes components are compiled using a version of Go, called goboring, which uses a FIPS-certified cryptographic module[1178] for all cryptographic functions.

Before provisioning DKP, you will need to follow your OS vendor's instructions to ensure that your OS, or OS images, are prepared for operating in FIPS mode. An example for RHEL is found here called Enabling FIPS mode[1179].

- Helpful topics:
  - https://csrc.nist.gov/publications/fips
  - FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION[1180]

> You cannot apply FIPS-mode to an existing cluster. You must create a new cluster with FIPS enabled. Similarly, a FIPS-mode cluster must remain a FIPS-mode cluster; you cannot change the cluster's FIPS status after you create it.

8.2.2 Infrastructure Requirements for FIPS-140-2 Mode

To ensure proper operations in FIPS mode, be sure that your environment meets these requirements:

FIPS 140 Mode Performance Impact (see page 1605)

8.2.2.1 Supported Operating Systems

Supported Operating Systems for FIPS mode are Red Hat Enterprise Linux and CentOS. See the Supported Operating Systems (see page 67) for details on the tested and supported versions.

8.2.3 Deploying a Cluster in FIPS mode

In order to create a cluster in FIPS mode, we must inform the bootstrap controllers of the appropriate image repository and version tags of the official D2iQ FIPS builds of Kubernetes.

1178 https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp3702.pdf
1179 https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/chap-federal_standards_and_regulations#sec-Enabling-FIPS-Mode
1180 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf

8.2.3.1 Supported FIPS Builds

Component	Repository	Version
Kubernetes	docker.io/mesosphere[1181]	v1.28.7+fips.0
etcd	docker.io/mesosphere[1182]	3.5.10+fips.0

AWS Example (see page 1190):

When creating a cluster, use the following command line options:

- `--ami <fips enabled AMI>` (AWS only)
- `--kubernetes-version <version>+fips.<build>`
- `--etcd-version <version>+fips.<build>`
- `--kubernetes-image-repository docker.io/mesosphere`
- `--etcd-image-repository docker.io/mesosphere`

For example:

```
dkp create cluster aws --cluster-name myFipsCluster \
--ami=ami-03dcaa75d45aca36f \
--kubernetes-version=v1.28.7+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
--etcd-image-repository=docker.io/mesosphere \
--etcd-version=3.5.10+fips.0
```

vSphere Example:

```
dkp create cluster vsphere \
  --cluster-name ${CLUSTER_NAME} \
  --network <NETWORK_NAME> \
  --control-plane-endpoint-host <xxx.yyy.zzz.000> \
  --data-center <DATACENTER_NAME> \
  --data-store <DATASTORE_NAME> \
  --folder <FOLDER_NAME> \
  --server <VCENTER_API_SERVER_URL> \
  --ssh-public-key-file <SSH_PUBLIC_KEY_FILE> \
  --resource-pool <RESOURE_POOL_NAME> \
```

1181 https://hub.docker.com/layers/mesosphere/kube-apiserver/v1.28.7_d2iq.0/images/sha256-1cafe40f5a17c86aa75574b25dd637bbeb377e2ac703532d72f1118c1e966e28?context=explore

1182 https://hub.docker.com/layers/mesosphere/etcd/v3.5.10_fips.0/images/sha256-60da8d0d3b37e71a4fa918ffa7ffd9963d9892b3ba43b8f63119d806f2e585ed?context=explore

```
--vm-template <TEMPLATE_NAME> \
--self-managed \
--kubernetes-version=v1.28.7+fips.0 \
--kubernetes-image-repository=docker.io/mesosphere \
--etcd-image-repository=docker.io/mesosphere --etcd-version=3.5.10+fips.0
```

8.2.4 Create FIPS 140 Images: Non-air-gapped Environments

8.2.4.1 Use Konvoy Image Builder to create images with FIPS-compliant binaries

8.2.4.2 Non-air-gapped Environment Create FIPS-140 images

KIB can produce images containing FIPS-140 compliant binaries. Use the `fips.yaml` override file (see page 1671) provided with the image bundles.

> You can also find these override files in the Konvoy Image Builder repo[1183].

8.2.4.2.1 Examples:

> The below snippets will create images with FIPS-compliant Kubernetes components. If you need the underlying OS to be FIPS-compliant, then you will need to provide the specific FIPS-compliant OS image, using the `--source-ami` flag for AWS.

- A non-air-gapped environment example of override file use is the command below, which produces a FIPS-compliant image on RHEL 8.4 for AWS:
  Replace `ami` with your infrastructure provisioner

```
konvoy-image build --overrides overrides/fips.yaml images/ami/rhel-84.yaml
```

- vSphere FIPS-complaint using `image.yaml` created during VM Template (see page 1654) configuration:

1183 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

```
konvoy-image build --overrides overrides/fips.yaml images/ova/<image.yaml>
```

In the Using Override Files with Konvoy Image Builder (see page 1670), there is a list of Override Files (see page 1671) and how to apply them.

8.2.5 Create FIPS 140 Images: Air-gapped Environment

KIB can produce images containing FIPS-140 compliant binaries. Use the `fips.yaml` override file (see page 1671) provided with the image bundles.

You can also find these override files in the Konvoy Image Builder repo[1184].

8.2.5.1 Examples:

The below snippets will create images with FIPS-compliant Kubernetes components. If you need the underlying OS to be FIPS-compliant, then you will need to provide the specific FIPS-compliant OS image, using the `--source-ami` flag for AWS.

- An air-gapped environment example of override file use is the command below which produces an AWS FIPS-compliant image on RHEL 8.4:

```
konvoy-image build --overrides offline-fips.yaml --overrides overrides/fips.yaml
images/ami/rhel-84.yaml
```

- vSphere FIPS-compliant air-gapped environment example:

```
konvoy-image build --overrides offline-fips.yaml --overrides overrides/fips.yaml
images/ova/<image.yaml>
```

1184 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

8.2.5.2 Pre-provisioned FIPS Infrastructure

If you are targeting a Pre-provisioned Installs (see page 148), you can create a FIPS-compliant cluster by doing the following:

1. Create a Pre-provisioned: Bootstrap Cluster (see page 1105)
2. Create a secret on the bootstrap cluster with the contents from `fips.yaml` override file[1185] and any other user overrides you wish to provide

```
kubectl create secret generic $CLUSTER_NAME-fips-overrides --from-
file=overrides.yaml=overrides.yaml
kubectl label secret $CLUSTER_NAME-fips-overrides clusterctl.cluster.x-k8s.io/move=
```

Here is a list of FIPS Override Files (see page 1671).

8.2.6 Validate FIPS in Cluster

You can use the FIPS validation tool to verify that specific components and services are FIPS-compliant. The tool checks the components by comparing their file signatures against ones stored in a signed signature file, and by checking that services are using the certified algorithms.

8.2.6.1 Run FIPS validation

To verify the cluster is FIPS compliant, run `dkp check cluster fips`. This command reads from the signature files embedded in the `dkp` executable in order to validate that specific components and services are FIPS-compliant. Run the command:

```
dkp check cluster fips
```

Upon successful completion, the command's output displays details about the deployment in JSON format. If validation fails, the output will say which components fail and a list of the nodes that failed validation will return.

The full command usage and flags include:

```
dkp check cluster fips [flags]
```

Flags:

1185 https://github.com/mesosphere/konvoy-image-builder/blob/main/overrides/fips.yaml

```
  -h, --help                          Help for fips
      --kubeconfig string             Path to the kubeconfig file for the fips
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
      --output-configmap string       ConfigMap to store result of the fips check.
(default "check-cluster-fips-output") (DEPRECATED: This flag will be removed in a
future release.)
      --signature-configmap string    ConfigMap with fips signature data to verify.
      --signature-file string         File containing fips signature data.
      --timeout duration              The length of time to wait before giving up.
Zero means wait forever (e.g. 1s, 2m, 3h). (default 10m0s)
```

8.2.6.1.1 Run FIPS validation with custom signature file

To validate FIPS-mode operation with the a custom signature file, you can use the `signature-file` flag, as in the following command. You also need to use the `signature-configmap` flag to set the name of the ConfigMap used to store your custom signature file.

```
dkp check cluster fips \
 --signature-file custom.json.asc \
 --signature-configmap custom-signature-file
```

8.2.6.1.2 Run FIPS validation with existing ConfigMap

If you already have a signature file stored in a ConfigMap, you can omit the `signature-file` flag, as in the following command:

```
dkp check cluster fips \
 --signature-configmap prod-rhel8-fips-signatures
```

8.2.6.2 Signature Files

The following signature files are already embedded in the `dkp` executable. They are provided for reference. You do not need to download them to run the FIPS check.

8.2.6.2.1 DKP Version 2.7.0

Operating System version	Kubernetes version	containerd version	Signature File URL
CentOS 7.9	v1.28.7	1.6.28	CentOS 7.9[1186]
Oracle 7.9	v1.28.7	1.6.28	Oracle 7.9[1187]
RHEL 7.9	v1.28.7	1.6.28	RHEL 7.9[1188]
RHEL 8.4	v1.28.7	1.6.28	RHEL 8.4[1189]
RHEL 8.6	v1.28.7	1.6.28	RHEL 8.6[1190]
RHEL 8.8	v1.28.7	1.6.28	RHEL 8.8[1191]

8.2.7 FIPS 140 Mode Performance Impact

8.2.7.1 Understand the performance impact from operating your cluster in FIPS 140 mode

The Go language cryptographic module, Goboring, relies on CGO's foreign function interface to call C-language functions exposed by the cryptographic module. Each call into the C library starts with a base overhead of 200ns.

One benchmark[1192] finds that the time to encrypt a single AES-128 block increased from 13ns to 209ns over the internal golang implementation. The preferred mode of D2iQ's FIPS module is `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`.

The aggregate impact on a stable control plane seems to be an increase of around 10% CPU utilization over default operation. Workloads that do not directly interact with the control plane are not affected.

1186 https://downloads.d2iq.com/dkp/fips/v2.8.1/manifest-centos-79.json.asc
1187 https://downloads.d2iq.com/dkp/fips/v2.8.1/manifest-oracle-79.json.asc
1188 https://downloads.d2iq.com/dkp/fips/v2.8.1/manifest-rhel-79.json.asc
1189 https://downloads.d2iq.com/dkp/fips/v2.8.1/manifest-rhel-84.json.asc
1190 https://downloads.d2iq.com/dkp/fips/v2.8.1/manifest-rhel-86.json.asc
1191 https://downloads.d2iq.com/dkp/fips/v2.8.1/manifest-rhel-88.json.asc
1192 https://github.com/golang/go/issues/21525

8.3 Registry Mirror Tools

Kubernetes does not natively provide a registry for hosting the container images you will use to run the applications you want to deploy on Kubernetes. Instead, Kubernetes requires you to use an external solution for storing and sharing container images. There are a variety of Kubernetes-compatible registry options that are compatible with DKP.

8.3.1 How Does it Work?

The **first time** you request an image from your local registry mirror, it pulls the image from the public registry (such as Docker) and stores it locally before handing it back to you. On subsequent requests, the local registry mirror is able to serve the image from its own storage.

8.3.2 Air-gapped vs Non-air-gapped Environments

In a non-air-gapped environment, you have access to the Internet. You retrieve artifacts from specialized repositories dedicated to them, such as Docker images contained in DockerHub and Helm Charts that come from a dedicated Helm Chart repository. You can also create your own local repository to hold the downloaded container images needed or any custom images you've created with the Konvoy Image Builder (see page 1626) tool.

In an **air-gapped environment**, you need a local repository to store Helm charts, Docker images, and other artifacts. Private registries provide security and privacy into enterprise container image storage, whether hosted remotely or on-premises locally in an air-gapped environment. DKP in an air-gapped environment **requires** a local container registry of trusted images to enable production-level Kubernetes cluster management. However, a local registry is an option in a non-air-gapped environment as well for speed and security.

If you want to use images from this local registry to deploy applications inside your Kubernetes cluster, you'll need to set up a **secret** for a private registry. The secret contains your login data, which Kubernetes needs to connect to your private repository.

8.3.3 Local Registry Tools Compatible with DKP

Tools such as JFrog™ Artifactory, Amazon® AWS ECR, Harbor™, and Nexus™ handle multiple types of artifacts in one local repository.

8.3.3.1 AWS ECR

AWS ECR (Elastic Container Registry) is supported as your air-gapped image registry or a non-air-gapped registry mirror. DKP added support for using AWS ECR as a default registry when uploading image bundles in AWS.

8.3.3.1.1 Prerequisites

- Ensure you have followed the steps to create proper permissions in AWS Minimal Permissions and Role to Create Clusters (see page 1156)
- Ensure you have created AWS Cluster IAM Policies, Roles, and Artifacts (see page 1162)

8.3.3.1.2 Upload the Air-gapped Image Bundle to the Local ECR Registry:

A cluster administrator uses DKP CLI commands to upload the image bundle to ECR with parameters:

```
dkp push bundle --bundle <bundle> --to-registry=<ecr-registry-address>/<ecr-registry-name>
```

Parameter definitions:

- `--bundle <bundle>` the group of images. The example below is for the DKP air-gapped environment bundle
- `--to-registry=<ecr-registry-address>/<ecr-registry-name>` to provide registry location for push

An **example** command would be:

```
dkp push bundle --bundle container-images/konvoy-image-bundle-v2.8.1.tar --to-registry=333000009999.dkr.ecr.us-west-2.amazonaws.com/can-test
```

NOTE: You can aslo set an environment variable with your registry address for **ECR**:

```
export REGISTRY_URL=<ecr-registry-URI>
```

- `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
- The environment where you are running the `dkp push` command must be authenticated with AWS in order to load your images into ECR.

8.3.3.1.3 Air-gapped Environment Information regarding your AWS ECR Account

The cluster administrator uses existing DKP CLI commands to create the cluster and refer to their internal ECR for image repository. The administrator does not need to provide static ECR registry credentials. See Use a Registry Mirror (see page 1609) and Create an EKS Cluster from the CLI (see page 1271) for more details.

8.3.3.2 JFrog Artifactory

JFrog Artifactory can function as a container registry, as well as an automated management tool for binaries and artifacts of all types. If you use **JFrog Artifactory**[1193] or **JFrog Container Registry**, you must update to a new version of the software. Use a build newer than version **7.11**; older versions are not compatible.

8.3.3.3 Nexus Registry

Nexus Repository[1194] is a package registry for your Docker images and Helm Chart repositories and supports Proxy, Hosted, and Group repositories. It can be used a single registry for all your Kubernetes deployments.

8.3.3.4 Harbor Registry

Install Harbor[1195] and configure any HTTP access required, as well as the system level parameters in the `harbor.yml` file. Then run the installer script. If you are upgrading from a previous version of Harbor, you update the configuration file and migrate your data to fit the database schema of the later version. For information about upgrading, see Upgrading Harbor[1196]. A version than **Harbor Registry v2.1.1**-5f52168e will support OCI images.

While seeding you may see error messages such as the following:

```
2023/09/12 20:01:18 retrying without mount: POST https://harbor-registry.daclusta/v2/
harbor-registry/mesosphere/kube-proxy/blobs/uploads/?from=mesosphere%2Fkube-
proxy&mount=sha256%3A9fd5070b83085808ed850ff84acc98a116e839cd5dcfefa12f2906b7d9c6e50d
&origin=REDACTED: UNAUTHORIZED: project not found, name: mesosphere: project not
found, name: mesosphere
```

This appears to indicate that the image was not successfully pushed to your Harbor docker registry, but it is a false positive error message. This will only affect version of the DKP binary newer than DKP 2.4.0. This does not affect any other Local Registry solution such as Nexus or Artifactory. You can safely ignore these error messages.

8.3.3.5 Bastion Host

If you have not set up a Bastion Host (see page 1068) yet, refer to that section of the documentation.

8.3.3.6 Related Topic:

If you need to **configure** a private registry with a registry mirror, see Use a Registry Mirror (see page 1609) for details on using that flag.

1193 https://jfrog.com/artifactory/
1194 https://www.nexusregistry.com/info/
1195 https://goharbor.io/docs/2.0.0/install-config/download-installer/
1196 https://goharbor.io/docs/2.0.0/administration/upgrade/

8.3.3.7 Next Topic:

Air-gapped Seed the Registry (see page 1611)

8.3.4 Use a Registry Mirror

Registry mirrors are local copies of images from a public registry that follows (or mirrors) the file structure of a public registry. You can push container images to a local registry from downloaded images or images that you create with the Konvoy Image Builder (see page 1626). If your environment allows Internet access, when an image is not found locally, the mirror registry would then consult its upstream registries. This kind of registry does not contain any images other than the ones requested.

8.3.4.1 Export Variables

Set the environment variable with your registry information.

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
export REGISTRY_CA=<path to the cacert file on the bastion>
```

Definitions:

- `REGISTRY_URL` : the address of an existing local registry accessible in the VPC that the new cluster nodes will be configured to use a mirror registry when pulling images.
    - EX: `https://registry.example.com`

Other local registries may use the options below:

- `REGISTRY_USERNAME` : optional-set to a user that has pull access to this registry.
- `REGISTRY_PASSWORD` : optional if username is not set.
- JFrog - `REGISTRY_CA` : (optional) the path on the bastion machine to the registry CA. This value is only needed if the registry is using a self-signed certificate and the AMIs are not already configured to trust this CA.
- To increase Docker Hub's rate limit[1197] use your Docker Hub credentials when creating the cluster, by setting flags `--registry-mirror-url=https://registry-1.docker.io --registry-mirror-username=<your-username> --registry-mirror-password=<your-password>` when running `dkp create cluster`.

1197 https://docs.docker.com/docker-hub/download-rate-limit/

8.3.4.2 Use Flags in Cluster Creation

If you set the `--registry-mirror` flag during cluster creation, the Kubelet will now send to requests to the dynamic-credential-provider with a different config. Only use one image registry per cluster.

To apply private registry configurations during the `dkp cluster create` operation, add the appropriate flags to the command:

Registry configuration	Flag
CA certificate chain to use while communicating with the registry mirror using Transport Layer Security(TLS)	`--registry-mirror-cacert file`
URL of a container registry to use as a mirror in the cluster	`--registry-mirror-url string` OR apply variable `${REGISTRY_URL}`
Set to a user that has pull access to this registry	`--registry-mirror-username string` OR apply variable `${REGISTRY_USERNAME}`
Password to authenticate the registry mirror	`--registry-mirror-password string` OR apply variable `${REGISTRY_PASSWORD}`

This is useful when using an internal registry and when Internet access is not available such as in an air-gapped environment. However, registry mirrors can be used in non-air-gapped environments as well for security and speed.

AWS ECR - Adding the mirror flags to EKS would enable new clusters to also use ECR as image mirror. If you set the `--registry-mirror` flag, the Kubelet will now send to requests to the `dynamic-credential-provider` with a different config. You can still pull your own images from ECR directly or use ECR as a mirror.

When the cluster is up and running, you can deploy and test workloads.

8.3.4.2.1 Registry Mirror Cluster Example

Selecting your provider, run:

```
dkp create cluster [aws, azure, gcp, preprovisoned, vsphere] \
    --cluster-name=${CLUSTER_NAME} \
    --registry-mirror-cacert /tmp/registry.pem \
    --registry-mirror-url=${REGISTRY_URL}
```

More information is found in the Custom Installation and Additional Infrastructure Tools (see page 1050) sections under the Create a New Cluster section of each Infrastructure Provider. Mirrors can be used in both air-gapped and non-air-gapped environments by adding the flag to the `dkp create cluster` command.

8.4 Air-gapped Seed the Registry

Before creating an air-gapped Kubernetes cluster, you need to load the required images in a local registry for the Konvoy component. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances (if deploying to AWS) or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, please refer to Local Registry Tools (see page 1606) page for more information.

1. If not already done in prerequisites, download (see page 76) the air-gapped bundle `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap cluster, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

   ```
   cd dkp-v2.8.1
   ```

3. Set an environment variable with your registry address and any other needed variables using this command:

   ```
   export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   export REGISTRY_CA=<path to the cacert file on the bastion>
   ```

4. Execute the following command to load the air-gapped image bundle into your private registry using any of the relevant flags to apply variables above:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --
to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-
registry-password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

You are now ready to create an air-gapped bootstrap cluster for a custom cluster for your infrastructure provider (see page 1050), or create an air-gapped cluster from the Day 1 - Basic Installs (see page 146) section for your provider.

EXAMPLE directory structure after extraction:

```
dkp-v2.8.1/
├── application-charts
│   ├── NOTICES.txt
│   ├── dkp-catalog-applications-charts-bundle-v2.8.1-rc.2.tar.gz
│   ├── dkp-insights-charts-bundle-v2.8.1-rc.2.tar.gz
│   └── dkp-kommander-charts-bundle-v2.8.1-rc.2.tar.gz
├── application-repositories
│   ├── dkp-catalog-applications-v2.8.1-rc.2.tar.gz
│   ├── dkp-insights-v2.8.1-rc.2.tar.gz
│   └── kommander-applications-v2.8.1-rc.2.tar.gz
├── container-images
│   ├── NOTICES.txt
│   ├── dkp-catalog-applications-image-bundle-v2.8.1-rc.2.tar
│   ├── dkp-insights-image-bundle-v2.8.1-rc.2.tar
│   ├── kommander-image-bundle-v2.8.1-rc.2.tar
│   └── konvoy-image-bundle-v2.8.1-rc.2.tar
├── dkp
├── kib
│   ├── LICENSE
│   ├── README.md
│   ├── ansible
│   ├── artifacts
│   ├── goss
│   ├── images
│   ├── konvoy-image
│   └── overrides
├── konvoy-bootstrap-image-v2.8.1-rc.2.tar
└── kubectl
```

8.5 Configure the Control Plane

Users can make modifications to the `KubeadmControlplane` cluster-api object to configure different kubelet options. Please see the following guide if you wish to configure your control plane beyond the existing options that are available from flags.

8.5.1 Prerequisites

Before you begin, make sure you have created your cluster using a bootstrap cluster from the respective Infrastructure Providers (see page 1050) section.

8.5.1.1 Modifying Audit Logs

In order to modify control plane option, get the appropriate `cluster-api` objects that describe the cluster by running the following command:

The following example uses AWS, but can be used for `gcp`, `azure`, `preprovisioned`, and `vsphere` clusters.

```
dkp create cluster aws -c {MY_CLUSTER_NAME} -o yaml --dry-run >>
{MY_CLUSTER_NAME}.yaml
```

When you open `{MY_CLUSTER_NAME}.yaml` with your favorite text editor, look for the `KubeadmControlPlane` object for your cluster. Below is an example object:

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: KubeadmControlPlane
metadata:
  name: my-cluster-control-plane
  namespace: default
spec:
  kubeadmConfigSpec:
    clusterConfiguration:
      apiServer:
        extraArgs:
          audit-log-maxage: "30"
          audit-log-maxbackup: "10"
          audit-log-maxsize: "100"
          audit-log-path: /var/log/audit/kube-apiserver-audit.log
```

```
          audit-policy-file: /etc/kubernetes/audit-policy/apiserver-audit-policy.yaml
          cloud-provider: aws
          encryption-provider-config: /etc/kubernetes/pki/encryption-config.yaml
        extraVolumes:
        - hostPath: /etc/kubernetes/audit-policy/
          mountPath: /etc/kubernetes/audit-policy/
          name: audit-policy
        - hostPath: /var/log/kubernetes/audit
          mountPath: /var/log/audit/
          name: audit-logs
      controllerManager:
        extraArgs:
          cloud-provider: aws
          configure-cloud-routes: "false"
      dns: {}
      etcd:
        local:
          imageTag: 3.5.7
      networking: {}
      scheduler: {}
    files:
    - content: |
        # Taken from https://github.com/kubernetes/kubernetes/blob/master/cluster/
gce/gci/configure-helper.sh
        # Recommended in Kubernetes docs
        apiVersion: audit.k8s.io/v1
        kind: Policy
        rules:
          # The following requests were manually identified as high-volume and low-
risk,
          # so drop them.
          - level: None
            users: ["system:kube-proxy"]
            verbs: ["watch"]
            resources:
              - group: "" # core
                resources: ["endpoints", "services", "services/status"]
          - level: None
            # Ingress controller reads 'configmaps/ingress-uid' through the unsecured
port.
            # TODO(#46983): Change this to the ingress controller service account.
            users: ["system:unsecured"]
            namespaces: ["kube-system"]
            verbs: ["get"]
            resources:
              - group: "" # core
                resources: ["configmaps"]
          - level: None
            users: ["kubelet"] # legacy kubelet identity
            verbs: ["get"]
            resources:
              - group: "" # core
```

```
            resources: ["nodes", "nodes/status"]
        - level: None
          userGroups: ["system:nodes"]
          verbs: ["get"]
          resources:
            - group: "" # core
              resources: ["nodes", "nodes/status"]
        - level: None
          users:
            - system:kube-controller-manager
            - system:kube-scheduler
            - system:serviceaccount:kube-system:endpoint-controller
          verbs: ["get", "update"]
          namespaces: ["kube-system"]
          resources:
            - group: "" # core
              resources: ["endpoints"]
        - level: None
          users: ["system:apiserver"]
          verbs: ["get"]
          resources:
            - group: "" # core
              resources: ["namespaces", "namespaces/status", "namespaces/finalize"]
        - level: None
          users: ["cluster-autoscaler"]
          verbs: ["get", "update"]
          namespaces: ["kube-system"]
          resources:
            - group: "" # core
              resources: ["configmaps", "endpoints"]
        # Don't log HPA fetching metrics.
        - level: None
          users:
            - system:kube-controller-manager
          verbs: ["get", "list"]
          resources:
            - group: "metrics.k8s.io"
        # Don't log these read-only URLs.
        - level: None
          nonResourceURLs:
            - /healthz*
            - /version
            - /swagger*
        # Don't log events requests.
        - level: None
          resources:
            - group: "" # core
              resources: ["events"]
        # node and pod status calls from nodes are high-volume and can be large,
don't log responses for expected updates from nodes
        - level: Request
```

```
            users: ["kubelet", "system:node-problem-detector",
"system:serviceaccount:kube-system:node-problem-detector"]
            verbs: ["update","patch"]
            resources:
              - group: "" # core
                resources: ["nodes/status", "pods/status"]
            omitStages:
              - "RequestReceived"
          - level: Request
            userGroups: ["system:nodes"]
            verbs: ["update","patch"]
            resources:
              - group: "" # core
                resources: ["nodes/status", "pods/status"]
            omitStages:
              - "RequestReceived"
          # deletecollection calls can be large, don't log responses for expected
namespace deletions
          - level: Request
            users: ["system:serviceaccount:kube-system:namespace-controller"]
            verbs: ["deletecollection"]
            omitStages:
              - "RequestReceived"
          # Secrets, ConfigMaps, and TokenReviews can contain sensitive & binary
data,
          # so only log at the Metadata level.
          - level: Metadata
            resources:
              - group: "" # core
                resources: ["secrets", "configmaps"]
              - group: authentication.k8s.io
                resources: ["tokenreviews"]
            omitStages:
              - "RequestReceived"
          # Get responses can be large; skip them.
          - level: Request
            verbs: ["get", "list", "watch"]
            resources:
              - group: "" # core
              - group: "admissionregistration.k8s.io"
              - group: "apiextensions.k8s.io"
              - group: "apiregistration.k8s.io"
              - group: "apps"
              - group: "authentication.k8s.io"
              - group: "authorization.k8s.io"
              - group: "autoscaling"
              - group: "batch"
              - group: "certificates.k8s.io"
              - group: "extensions"
              - group: "metrics.k8s.io"
              - group: "networking.k8s.io"
              - group: "node.k8s.io"
```

```
            - group: "policy"
            - group: "rbac.authorization.k8s.io"
            - group: "scheduling.k8s.io"
            - group: "settings.k8s.io"
            - group: "storage.k8s.io"
          omitStages:
            - "RequestReceived"
        # Default level for known APIs
        - level: RequestResponse
          resources:
            - group: "" # core
            - group: "admissionregistration.k8s.io"
            - group: "apiextensions.k8s.io"
            - group: "apiregistration.k8s.io"
            - group: "apps"
            - group: "authentication.k8s.io"
            - group: "authorization.k8s.io"
            - group: "autoscaling"
            - group: "batch"
            - group: "certificates.k8s.io"
            - group: "extensions"
            - group: "metrics.k8s.io"
            - group: "networking.k8s.io"
            - group: "node.k8s.io"
            - group: "policy"
            - group: "rbac.authorization.k8s.io"
            - group: "scheduling.k8s.io"
            - group: "settings.k8s.io"
            - group: "storage.k8s.io"
          omitStages:
            - "RequestReceived"
        # Default level for all other requests.
        - level: Metadata
          omitStages:
            - "RequestReceived"
    path: /etc/kubernetes/audit-policy/apiserver-audit-policy.yaml
    permissions: "0600"
  - content: |
      #!/bin/bash
      # CAPI does not expose an API to modify KubeProxyConfiguration
      # this is a workaround to use a script with preKubeadmCommand to modify the
kubeadm config files
      # https://github.com/kubernetes-sigs/cluster-api/issues/4512
      for i in $(ls /run/kubeadm/ | grep 'kubeadm.yaml\|kubeadm-join-config.yaml');
do
        cat <<EOF>> "/run/kubeadm//$i"
      ---
      kind: KubeProxyConfiguration
      apiVersion: kubeproxy.config.k8s.io/v1alpha1
      metricsBindAddress: "0.0.0.0:10249"
      EOF
      done
```

```
      path: /run/kubeadm/konvoy-set-kube-proxy-configuration.sh
      permissions: "0700"
    - content: |
        [metrics]
          address = "0.0.0.0:1338"
          grpc_histogram = false
      path: /etc/containerd/conf.d/konvoy-metrics.toml
      permissions: "0644"
    - content: |
        #!/bin/bash
        systemctl restart containerd

        SECONDS=0
        until crictl info
        do
          if (( SECONDS > 60 ))
          then
             echo "Containerd is not running. Giving up..."
             exit 1
          fi
          echo "Containerd is not running yet. Waiting..."
          sleep 5
        done
      path: /run/konvoy/restart-containerd-and-wait.sh
      permissions: "0700"
    - contentFrom:
        secret:
          key: value
          name: my-cluster-etcd-encryption-config
      owner: root:root
      path: /etc/kubernetes/pki/encryption-config.yaml
      permissions: "0640"
    format: cloud-config
    initConfiguration:
      localAPIEndpoint: {}
      nodeRegistration:
        kubeletExtraArgs:
          cloud-provider: aws
        name: '{{ ds.meta_data.local_hostname }}'
    joinConfiguration:
      discovery: {}
      nodeRegistration:
        kubeletExtraArgs:
          cloud-provider: aws
        name: '{{ ds.meta_data.local_hostname }}'
    preKubeadmCommands:
    - systemctl daemon-reload
    - /run/konvoy/restart-containerd-and-wait.sh
    - /run/kubeadm/konvoy-set-kube-proxy-configuration.sh
  machineTemplate:
    infrastructureRef:
      apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
```

```
      kind: AWSMachineTemplate
      name: my-cluster-control-plane
      namespace: default
    metadata: {}
  replicas: 3
  rolloutStrategy:
    rollingUpdate:
      maxSurge: 1
    type: RollingUpdate
  version: v1.28.7
```

If you use the above example as-is, ensure you update your Kubeneters version number on the final line by replacing the x.

Now a user can configure the fields below for the log backend. The log backend will write audit events to a file in JSON[1198] format. You can configure the log audit backend using the `kube-apiserver` flags shown below:

```
audit-log-maxage
audit-log-maxbackup
audit-log-maxsize
audit-log-path
```

See upstream documentation[1199]for more information.

After modifying the values appropriately, you can create the cluster by running the command below:

```
kubectl create -f {MY_CLUSTER_NAME}.yaml
```

Once the cluster is created, users can get the corresponding kubeconfig for the cluster by running the following command:

```
dkp get kubeconfig -c {MY_CLUSTER_NAME} >> {MY_CLUSTER_NAME}.conf
```

1198 https://jsonlines.org/
1199 https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/#log-backend

8.5.1.2 Viewing the Audit Logs

Fluent Bit is disabled on the management cluster by default, to view the audit logs run the command below:

```
dkp diagnose --kubeconfig={MY_CLUSTER_NAME}.conf
```

A file similar to `support-bundle-2022-08-15T02_28_48.tar.gz` will be created. Untar the file using a command similar to the example below:

```
tar -xzf support-bundle-2022-08-15T02_28_48.tar.gz
```

Navigate to the node-diagnostics sub directory from the extracted file with a command like the one shown below:

```
cd support-bundle-2022-08-15T02_28_48/node-diagnostics
```

Finally, to find the audit logs run the following command:

```
$ find . -type f | grep audit.log
./ip-10-0-142-117.us-west-2.compute.internal/data/kube_apiserver_audit.log
./ip-10-0-148-139.us-west-2.compute.internal/data/kube_apiserver_audit.log
./ip-10-0-128-181.us-west-2.compute.internal/data/kube_apiserver_audit.log
```

See other related pages below:

Fluent bit (see page 952)

8.6 Update Cluster Nodepools

Upgrading a node pool involves draining the existing nodes in the node pool and replacing them with new nodes. In order to ensure minimum downtime and maintain high availability of the critical application workloads during the upgrade process, we recommend deploying Pod Disruption Budget (Disruptions[1200]) for your critical applications.

The Pod Disruption Budget will prevent any impact on critical applications as a result of misconfiguration or failures during the upgrade process.

1200 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/

8.6.1 Prerequisites:

- Deploy Pod Disruption Budget[1201] (PDB)
- Konvoy Image Builder (see page 1626) (KIB)

8.6.2 Steps:

1. Deploy Pod Disruption Budget for your critical applications. If your application can tolerate only one replica to be unavailable at a time, then you can set Pod disruption budget as shown in the following example. The example below is for NVIDIA GPU node pools, but the process is the same for all node pools.

> Repeat this step for each additional node pool.

Create the file: `pod-disruption-budget-nvidia.yaml`

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: nvidia-critical-app
spec:
  maxUnavailable: 1
  selector:
    matchLabels:
      app: nvidia-critical-app
```

Apply the YAML file above using the following command:

```
kubectl create -f pod-disruption-budget-nvidia.yaml
```

2. Prepare OS image for your node pool using Konvoy Image Builder (see page 1626).

8.6.3 Related Topics:

Upgrade Konvoy (see page 1745)

1201 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/

8.7 Verify your Cluster and DKP Installation

Check DKP components to verify the status of your cluster

This section contains information on how to verify a DKP installation.

8.7.1 Check the Cluster Infrastructure and Nodes

DKP ships with some default diagnosis tools to check your cluster, such as the `describe` command. You can use those tools to validate your installation.

If you have not done so already, set the environment variable for your cluster name, substituting `dkp-example` with the name of your cluster:

```
export CLUSTER_NAME=dkp-example
```

Then, run this command to check the health of the cluster infrastructure:

```
dkp describe cluster --cluster-name=${CLUSTER_NAME}
```

For more details on the `dkp describe cluster` command, refer to dkp describe cluster (see page 1892).

A healthy cluster returns an output similar to:

```
NAME                                                        READY  SEVERITY
REASON  SINCE  MESSAGE
Cluster/dkp-example                                         True
121m
├─ClusterInfrastructure - AWSCluster/dkp-example            True
121m
├─ControlPlane - KubeadmControlPlane/dkp-example-control-plane  True
121m
│ ├─Machine/dkp-example-control-plane-h52t6                 True
121m
│ ├─Machine/dkp-example-control-plane-knrrh                 True
121m
│ └─Machine/dkp-example-control-plane-zmjjx                 True
121m
└─Workers
```

```
  └─MachineDeployment/dkp-example-md-0                          True
121m
    ├─Machine/dkp-example-md-0-88488cb74-2vxjq                  True
121m
    ├─Machine/dkp-example-md-0-88488cb74-84xsd                  True
121m
    ├─Machine/dkp-example-md-0-88488cb74-9xmc6                  True
121m
    └─Machine/dkp-example-md-0-88488cb74-mjf6s                  True
121m
```

Use this `kubectl` command to check to see if all cluster nodes are ready:

```
kubectl get nodes
```

The output should look similar to this, with all statuses set to `Ready`

```
NAME                                         STATUS   ROLES                  AGE
VERSION
ip-10-0-112-116.us-west-2.compute.internal   Ready    <none>                 135m
v1.21.6
ip-10-0-122-142.us-west-2.compute.internal   Ready    <none>                 135m
v1.21.6
ip-10-0-186-214.us-west-2.compute.internal   Ready    control-plane,master   133m
v1.21.6
ip-10-0-231-82.us-west-2.compute.internal    Ready    control-plane,master   135m
v1.21.6
ip-10-0-71-114.us-west-2.compute.internal    Ready    <none>                 135m
v1.21.6
ip-10-0-71-207.us-west-2.compute.internal    Ready    <none>                 135m
v1.21.6
ip-10-0-85-253.us-west-2.compute.internal    Ready    control-plane,master   137m
v1.21.6
```

To verify a successful installation, all of the previous commands should return as Ready or True.

8.7.2 Monitor the CAPI resources

Obtain a list of all resources that comprise a cluster and their statuses:

```
kubectl get cluster-api
```

8.7.3 Verify all Pods

All pods installed by DKP should be in `Running` or `Completed` status if the install is successful.

```
kubectl get pods --all-namespaces
```

8.7.4 Troubleshooting

If any pod is not in `Running` or `Completed` status, you need to investigate further. If it appears that something has not deployed properly or fully, run a diagnostic bundle (see page 1758):

```
dkp diagnose
```

This collects information from pods and infrastructure. For more information, see more about generating a support bundle and the different configurations that it supports (see page 1758).

8.8 GPU for Konvoy

In this version of DKP, the nodes with NVIDIA GPUs are configured with nvidia-gpu-operator (Overview — NVIDIA Cloud Native Technologies documentation[1202]) and NVIDIA drivers to support the container runtime.

This page will link you to all the relevant GPU pages necessary such as Updating Cluster Nodepools (see page 1620) or KIB for GPU (see page 1649). The remainder of GPU information (see page 998) is found in Day 2 Operations under Kommander component section of documentation.

8.9 Delete a DKP Cluster with One Command

You can use a single command line entry to delete a Kubernetes cluster on any of the platforms supported by DKP. Deleting a cluster means removing the cluster, all of its nodes, and all of the platform applications that were deployed on it as part of its creation. Use this command with extreme care, as it is not reversible!

You need to delete the attachment for any clusters attached in Kommander before running the `delete` command.

Persistent Volumes (PVs) are not deleted automatically by design in order to preserve your data. However, they take up storage space if not deleted. You must delete PVs manually. Information for backup of a cluster and PVs is on the page in documentation called Back up your Cluster's Applications and Persistent Volumes (see page 863) .

1202 https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/overview.html

With Vsphere clusters, `dkp delete` doesn't delete the virtual disks backing the PVs for DKP add ons. Therefore internal VMware cluster runs out of storage eventually. Theses PVs are only visible if VSAN is installed which gives users a Container Native Storage tab.

Set the environment variable to be used throughout this documentation:

```
export CLUSTER_NAME=cluster-example
```

The basic DKP `delete` command structure is:

```
dkp delete cluster --cluster-name=${CLUSTER_NAME} --self-managed --kubeconfig=$
{CLUSTER_NAME}.conf
```

When you use the `--self-managed` flag, the prerequisite components and resources are moved from the self-managed cluster before deleting. When you omit this flag (the default value is false) the resources are assumed to be installed in a management cluster. The default value is false, or no flag.

This command performs the following actions:

- Creates a local bootstrap cluster
- Moves controllers to it
- Deletes the management cluster
- Deletes the local bootstrap cluster

9 Konvoy Image Builder

Konvoy Image Builder (KIB) is a complete solution for building Cluster API compliant images. The goal of Konvoy Image Builder is to produce a common operating surface to run Konvoy across heterogeneous infrastructure. KIB relies on:

- **Ansible** to install software, configure, and sanitize systems for running Konvoy.
- **Packer** is used to build images for cloud environments.
- **Goss** is used to validate systems are capable of running Konvoy.

This section describes how to use KIB to create a Cluster API[1203] compliant machine images. Machine images contain configuration information[1204] and software to create a specific, pre-configured, operating environment. For example, you can create an image of your current computer system settings and software. The machine image can then be replicated and distributed, creating your computer system for other users. KIB uses variable overrides (see page 1670) to specify base image and container images to use in your new machine image. The variable overrides files for Nvidia and FIPS can be ignored unless adding an overlay feature.

9.1 How KIB Works

You can use KIB to build machine images, but first you need to be aware of the default behaviors of KIB. Stated very simply, KIB installs `kubeadm` and the other basic components you need, so that when the machine boots for the first time, it becomes a Kubernetes control plane or worker node, and then can form or join a cluster.

KIB does this by booting a computer using a stripped-down base image, like an AMI, and then runs a series of steps to install all of the components that DKP needs. When the installation completes, KIB takes a snapshot or backup of that machine image and saves it. This becomes the image or AMI, and so on, that you use when building the cluster.

9.2 Prerequisites

Before you begin, you must ensure your versions of KIB and DKP are compatible:

- Download the Konvoy Image Builder bundle from the KIB Version column of the chart below for your version of DKP prefixed with `konvoy-image-bundle` for your Operating System.
- Check the Supported Infrastructure Operating Systems (see page 67) and the Supported Kubernetes Version (see page 1706) for your Provider.
- An x86_64-based Linux or MacOS machine
- A Container engine installed:

1203 https://cluster-api.sigs.k8s.io/
1204 https://github.com/mesosphere/konvoy-image-builder/tree/main/images

    - Version Docker®[1205] container engine version 18.09.2 or higher installed for Linux or MacOS - On macOS, Docker runs in a virtual machine which needs configured with at least 8 GB of memory.
    - Version 4.0 of Podman[1206] or higher for Linux.
- For air-gapped only - a local registry (see page 1606)

9.2.1 Additional Configurations to Know using KIB

A variety of flags can be used to pass variables:

- **Using Override files**
    - You can use overrides files (see page 1670) to customize some of the components installed on this machine image. The KIB base override files are located in this Github repository[1207].
- **Customize image YAML**
    - Begin creating an image, interrupt the process so that the manifest.jason gets built and you can open and edit keys in the YAML. Instructions are located inCustomize your Image YAML or Manifest File (see page 1661).
- **Using HTTP/S Proxies**
    - In some networked environments, the machines used for building images can reach the Internet, but only through an HTTP/S proxy. For DKP to operate in these networks, you need a way to specify what proxies to use (see page 1053). Further explanation is found in Using HTTP/S Proxy with KIB Images (see page 1681)

9.2.2 Compatible DKP to KIB Versions

Along with the KIB Bundle, we publish a file containing checksums for the bundle files. The recommendation for using these checksums is to verify the integrity of the downloads.

- On the corresponding link page, download the package prefixed with `konvoy-image-bundle` for your OS.

DKP Version	KIB Version (Click for bundle download)
v2.8.0	v2.9.7[1208]
v2.7.1	v2.8.7[1209]

1205 https://docs.docker.com/get-docker/
1206 https://podman.io/getting-started/installation
1207 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides
1208 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.9.7
1209 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.8.7

DKP Version	KIB Version (Click for bundle download)
v2.7.0	v2.8.6[1210]
v2.6.2	v2.5.5[1211]
v2.6.1	v2.5.4[1212]
v2.6.0	v2.5.0[1213]
v2.5.2	v2.2.13[1214]
v2.5.1	v2.2.8[1215]
v2.5.0	v2.2.6[1216]

9.2.3 Extract KIB Bundle

Extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION` folder. The bundled version of `konvoy-image` contains an embedded `docker` image that contains all the requirements for building.

> The `konvoy-image` binary and all supporting folders are also extracted and bind mount[1217] places the current working directory (`${PWD}`) into the container to be used.

- Set environment variables for AWS access[1218]. The following variables must be set using your credentials including required IAM (see page 1629):

1210 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.8.6
1211 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.5.5
1212 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.5.4
1213 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.5.0
1214 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.2.13
1215 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.2.8
1216 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.2.6
1217 https://docs.docker.com/storage/bind-mounts/
1218 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-envvars.html

```
export AWS_ACCESS_KEY_ID
export AWS_SECRET_ACCESS_KEY
export AWS_DEFAULT_REGION
```

9.2.4 Next Steps:

Either return to Basic Install (see page 146) or Custom Install (see page 1050) instructions, or for more KIB specific provider information you can continue to the provider link below for additional information:

- Using KIB with AWS (see page 1629)
- Using KIB with Azure (see page 1642)
- Using KIB with GCP (see page 1646)
- Using KIB with GPU (see page 1649)
- Using KIB with vSphere (see page 1652)
- Using KIB with Pre-provisioned Environments (see page 1660)
- Customize your Image (see page 1661)
- Konvoy Image Builder CLI (see page 1683)

9.2.5 Return:

If using the Day 1 - Basic Installs by Infrastructure (see page 146) instructions, proceed (or return) to that section to install and setup DKP based on your infrastructure environment provider.

If using the Custom Installation and Additional Infrastructure Tools (see page 1050) instructions, proceed (or return) to that section and select the infrastructure provider you are using.

9.3 Using KIB with AWS

The following section describes how to use Konvoy Image Builder (KIB) with Amazon Web Services (AWS). A minimal set of permissions from the AWS Image Builder Book[1219] should be met before beginning.

Section Contents

- Minimal IAM Permissions for KIB (see page 1629)
- AWS Image Integrated with DKP CLI (see page 1632)
- Create a Custom AMI (see page 1634)
- KIB for EKS (see page 1642)

9.3.1 Minimal IAM Permissions for KIB

Configure IAM Prerequisites before building an AWS Image

This section guides you in creating and using a minimally-scoped policy to create an Image for an AWS account using Konvoy Image Builder.

1219 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis

9.3.1.1 Prerequisites

Before applying the IAM Policies, verify the following:

- You have a valid AWS account with credentials configured[1220] that can manage CloudFormation Stacks, IAM Policies, IAM Roles, and IAM Instance Profiles.
- The AWS CLI utility[1221] is installed.

9.3.1.1.1 Minimal Permissions

The following is an AWSCloudformation stack that creates the minimal policy to run KIB in AWS.

1. Copy the following contents into a file:

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  AWSIAMInstanceKIBUser:
    Properties:
      InstanceProfileName: KIBUserInstnaceProfile
      Roles:
        - Ref: KIBUserRole
    Type: AWS::IAM::InstanceProfile
  AWSIAMManagedPolicyKIBPolicy:
    Properties:
      Description: Minimal policy to run KIB in AWS
      ManagedPolicyName: kib-policy
      PolicyDocument:
        Statement:
          - Action:
              - ec2:AssociateRouteTable
              - ec2:AssociateRouteTable
              - ec2:AttachInternetGateway
              - ec2:AttachVolume
              - ec2:AuthorizeSecurityGroupIngress
              - ec2:CreateImage
              - ec2:CreateInternetGateway
              - ec2:CreateKeyPair
              - ec2:CreateRoute
              - ec2:CreateRouteTable
              - ec2:CreateSecurityGroup
              - ec2:CreateSubnet
              - ec2:CreateTags
              - ec2:CreateVolume
              - ec2:CreateVpc
              - ec2:DeleteInternetGateway
```

1220 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html
1221 https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html

```
              - ec2:DeleteKeyPair
              - ec2:DeleteRouteTable
              - ec2:DeleteSecurityGroup
              - ec2:DeleteSnapshot
              - ec2:DeleteSubnet
              - ec2:DeleteVolume
              - ec2:DeleteVpc
              - ec2:DeregisterImage
              - ec2:DescribeAccountAttributes
              - ec2:DescribeImages
              - ec2:DescribeInstances
              - ec2:DescribeInternetGateways
              - ec2:DescribeKeyPairs
              - ec2:DescribeNetworkAcls
              - ec2:DescribeNetworkInterfaces
              - ec2:DescribeRegions
              - ec2:DescribeRouteTables
              - ec2:DescribeSecurityGroups
              - ec2:DescribeSubnets
              - ec2:DescribeVolume
              - ec2:DescribeVpcAttribute
              - ec2:DescribeVpcClassicLink
              - ec2:DescribeVpcClassicLinkDnsSupport
              - ec2:DescribeVpcs
              - ec2:DetachInternetGateway
              - ec2:DetachVolume
              - ec2:DisassociateRouteTable
              - ec2:ModifyImageAttribute
              - ec2:ModifySnapshotAttribute
              - ec2:ModifySubnetAttribute
              - ec2:ModifyVpcAttribute
              - ec2:RegisterImage
              - ec2:RevokeSecurityGroupEgress
              - ec2:RunInstances
              - ec2:StopInstances
              - ec2:TerminateInstances
            Effect: Allow
            Resource:
              - '*'
        Version: 2012-10-17
      Roles:
        - Ref: KIBUserRole
    Type: AWS::IAM::ManagedPolicy
    Version: 2012-10-17
  KIBUserRole:
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action:
              - sts:AssumeRole
            Effect: Allow
            Principal:
```

```
            Service:
              - ec2.amazonaws.com
        - Action:
            - sts:AssumeRole
          Effect: Allow
          Principal:
            AWS: arn:aws:iam::MYAWSACCOUNTID:root
      Version: 2012-10-17
    RoleName: kib-user-role
  Type: AWS::IAM::Role
```

2. Replace the following with the correct values:

- `MYFILENAME.yaml` - give your file a meaningful name.
- `MYSTACKNAME` - give your cloudformation stack a meaningful name.

3. Run the following command to create the stack:

```
aws cloudformation create-stack --template-body=file://MYFILENAME.yaml --stack-
name=MYSTACKNAME --capabilities  CAPABILITY_NAMED_IAM
```

9.3.1.1.2 Next Step:

AWS Image Integrated with DKP CLI (see page 1632)

9.3.2 AWS Image Integrated with DKP CLI

This procedure describes how to create a Cluster API[1222] compliant Amazon Machine Image (AMI).

A customized image requires the Konvoy Image Builder (see page 1626) tool to be downloaded (see page 76) and use variable overrides (see page 1670) to specify the base image and container images to use in your new AMI. To create a custom AMI and take advantage of enhanced cluster operations, explore the Using KIB with AWS (see page 1629) topics for more options.

In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1634). A customized image requires the Konvoy Image Builder (see page 1626) tool to be downloaded (see page 76) where you can use variable overrides (see page 1670) to specify the base image and container images for use in your new custom AMI (see page 1634).

1222 https://cluster-api.sigs.k8s.io/

9.3.2.1 Prerequisites

Before you begin, you must:

- Check the Supported Infrastructure Operating Systems (see page 67)
- Check the Supported Kubernetes Version (see page 1706) for your Provider.
- Create a working `Docker` or other Registry setup.
- Ensure you have met the minimal set of permissions from the AWS Image Builder Book[1223].

9.3.2.1.1 Build the Image

Depending on which version of DKP (see page 1626) you are running, steps and flags will be different.

9.3.2.1.1.1 Execute the following to begin image creation:

Run the `dkp-image-builder create` command to build and validate the image.

```
dkp-image-builder create image aws/centos-79.yaml
```

By default it builds in the `us-west-2` region. to specify another region set the `--region` flag:

```
dkp-image-builder create image aws --region us-east-1 images/ami/centos-79.yaml
```

Once DKP provisions the image successfully, the `ami` id is printed and written to the `packer.pkr.hcl` file. This file has an `artifact_id` field whose value provides the name of the AMI ID as shown in the example below:

```
{
      "name": "rhel-7.9-fips",
      "builder_type": "amazon-ebs",
      "build_time": 1659486130,
      "files": null,
      "artifact_id": "us-west-2:ami-0f2ef742482e1b829",
      "packer_run_uuid": "0ca500d9-a5f0-815c-6f12-aceb4d46645b",
      "custom_data": {
        "containerd_version": "",
        "distribution": "RHEL",
        "distribution_version": "7.9",
        "kubernetes_cni_version": "",
        "kubernetes_version": "1.24.5+fips.0"
      }
```

1223 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis

```
    }
```

9.3.2.1.2 Next Step:

Create a Custom AMI (see page 1634)

9.3.3 Create a Custom AMI

9.3.3.1 Learn how to build a custom AMI for use with DKP

AMI images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create an AMI image of your current computer system settings and software. The AMI image can then be replicated and distributed, creating your computer system for other users. You can use overrides files to customize some of the components installed on your machine image. For example, you could tell KIB to install the FIPS versions of the Kubernetes components.

This procedure describes how to use the Konvoy Image Builder (see page 1626) (KIB) to create a Cluster API[1224] compliant Amazon Machine Image (AMI). KIB uses variable overrides (see page 1670) to specify base image and container images to use in your new AMI.

> ⚠ In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1629). Explore the Customize your Image (see page 1661) topic for more options about overrides.

9.3.3.2 Prerequisites

Before you begin, you must:

- Download the KIB (see page 0) bundle for your version of DKP prefixed with `konvoy-image-bundle` for your OS.
- Check the Supported Infrastructure Operating Systems (see page 67)
- Check the Supported Kubernetes Version (see page 1706) for your Provider.
- Create a working registry:
    - Version 4.0 of Podman[1225] or higher for Linux. Host requirements found here: https://kind.sigs.k8s.io/docs/user/rootless/#host-requirements
    - Version 20.10.0 of Docker[1226] or higher for Linux or MacOS

1224 https://cluster-api.sigs.k8s.io/
1225 https://podman.io/getting-started/installation
1226 https://www.docker.com/

- Ensure you have met the minimal set of permissions from the AWS Image Builder Book[1227].
- A Minimal IAM Permissions for KIB (see page 1629) to create an Image for an AWS account using Konvoy Image Builder.

9.3.3.2.1 Extract KIB Bundle

If not done previously during Konvoy Image Builder download in Prerequisites, extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION` folder. **Otherwise, proceed to Build the Image.**

In previous DKP releases, the distro package bundles were included in the downloaded air-gapped bundle. Currently, that air-gapped bundle contains the following artifacts with the exception of the distro packages:

- DKP Kubernetes packages
- Python packages (provided by upstream)
- Containerd tarball

1. Download (see page 76) `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz && cd dkp-v2.8.0/kib
   ```

2. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.
3. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle` . This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml` . Example command:

   ```
   ./konvoy-image create-package-bundle --os redhat-8.4 --output-
   directory=artifacts
   ```

   NOTE: For FIPS, pass the flag: `--fips`

   NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY` . Example command:

   ```
   export RHSM_ACTIVATION_KEY="-ci"
   export RHSM_ORG_ID="1232131"
   ```

4. Follow the instructions below to build an AMI.

1227 https://image-builder.sigs.k8s.io/capi/providers/aws.html#required-permissions-to-build-the-aws-amis

The `konvoy-image` binary and all supporting folders are also extracted. When run, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

- Set environment variables for AWS access[1228]. The following variables must be set using your credentials including required IAM (see page 1629):

```
export AWS_ACCESS_KEY_ID
export AWS_SECRET_ACCESS_KEY
export AWS_DEFAULT_REGION
```

- If you have an override file (see page 1670) to configure specific attributes of your AMI file, add it. Instructions for customizing an override file are found on this page: Image Overrides (see page 1678)

9.3.3.3 Build the Image

Depending on which version of DKP (see page 1626) you are running, steps and flags will be different. To deploy in a region where CAPI images are not provided, you need to use KIB to create your own image for the region. For a list of supported AWS regions, refer to the Published AMI[1229]information from AWS.

9.3.3.3.1 Begin image creation:

Run the `konvoy-image` command to build and validate the image.

```
konvoy-image build aws images/ami/rhel-86.yaml
```

By default, it builds in the `us-west-2` region. to specify another region set the `--region` flag as shown in the command below:

```
konvoy-image build aws --region us-east-1 images/ami/rhel-86.yaml
```

Ensure you have named the correct AMI image YAML file for your OS[1230] in the `konvoy-image build` command.

1228 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-envvars.html
1229 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html
1230 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ami

After KIB provisions the image successfully, the `ami` id is printed and written to the `packer.pkr.hcl` (Packer config) file. This file has an `artifact_id` field whose value provides the name of the AMI ID as shown in the example below. That is the `ami` you use in the `dkp create cluster` command:


```
{
  "builds": [
    {
      "name": "kib_image",
      "builder_type": "amazon-ebs",
      "build_time": 1698086886,
      "files": null,
      "artifact_id": "us-west-2:ami-04b8dfef8bd33a016",
      "packer_run_uuid": "80f8296c-e975-d394-45f9-49ef2ccc6e05",
      "custom_data": {
        "containerd_version": "",
        "distribution": "RHEL",
        "distribution_version": "8.6",
        "kubernetes_cni_version": "",
        "kubernetes_version": "1.26.6"
      }
    }
  ],
  "last_run_uuid": "80f8296c-e975-d394-45f9-49ef2ccc6e05"
}
```


To use a custom AMI when creating your cluster (see page 1190), you must create that AMI using KIB first. Then perform the export and name the custom AMI for use in the command `dkp create cluster`:

```
export AWS_AMI_ID=ami-<ami-id-here>
```

Inside the sections for either Non-air-gapped (see page 1190) or Air-gapped (see page 1218) cluster creation, you will find the instructions for how to apply custom images.

9.3.3.3.2 Related Information

For information on related topics or procedures, refer to the following:

- Creating GPU enabled on-premises configurations (see page 1249)

9.3.3.4 AWS Air-gapped AMI

To create an Image using Konvoy Image Builder (KIB) for use in an air-gapped cluster, follow these instructions. AMI images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create an AMI image of your current computer system settings and software. The AMI image can then be replicated and distributed, creating your computer system for other users. You can use overrides files to customize some of the components installed on your machine image. For example, you could tell KIB to install the FIPS versions of the Kubernetes components.

9.3.3.4.1 Prerequisites

- Minimal IAM Permissions for KIB (see page 1629) to create an Image for an AWS account using Konvoy Image Builder.

> ⚠ In previous DKP releases, AMI images provided by the upstream CAPA project would be used if you did not specify an AMI. However, the upstream images are not recommended for production and may not always be available. Therefore, DKP now requires you to specify an AMI when creating a cluster. To create an AMI, use Konvoy Image Builder (see page 1626).
> Explore the Customize your Image (see page 1661) topic for more options.
> Using KIB (see page 1626), you can build an AMI without requiring access to the internet by providing an additional `--override` flag.

In previous DKP releases, the distro package bundles were included in the downloaded air-gapped bundle. Currently, that air-gapped bundle contains the following artifacts with the exception of the distro packages:

- DKP Kubernetes packages
- Python packages (provided by upstream)
- Containerd tarball

1. Download (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz && cd dkp-v2.8.1/kib
   ```

2. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.
3. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle` . This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml` . Example command:

   ```
   ./konvoy-image create-package-bundle --os redhat-8.4 --output-
   directory=artifacts
   ```

   NOTE: For FIPS, pass the flag: `--fips`

   NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY` . Example command:

```
export RHSM_ACTIVATION_KEY="-ci"
export RHSM_ORG_ID="1232131"
```

4. Follow the instructions below to build an AMI.

Depending on which version of DKP (see page 1626) you are running, steps and flags will be different. To deploy in a region where CAPI images are not provided, you need to use KIB to create your own image for the region. For a list of supported AWS regions, refer to the Published AMI[1231] information from AWS.

9.3.3.4.1.1 Run the following to begin image creation:

5. Run the `konvoy-image` command to build and validate the image. Ensure you have named the correct AMI image YAML file for your OS[1232] in the `konvoy-image build` command.

```
./konvoy-image build aws images/ami/centos-79.yaml --overrides overrides/
offline.yaml
```

By default, it builds in the `us-west-2` region. **To specify another region** set the `--region` flag as shown in the example command below:

```
./konvoy-image build aws --region us-east-1 images/ami/centos-79.yaml --overrides
overrides/offline.yaml
```

Instructions for customizing an override file are found on this page: Image Overrides (see page 1678)

After KIB provisions the image successfully, the `ami` id is printed and written to the `packer.pkr.hcl` (Packer config) file. This file has an `amazon-ebs.kib_image` field whose value provides the name of the AMI ID as shown in the example below. That is the `ami` you use in the `dkp create cluster` command:

```
...
    amazon-ebs.kib_image: Adding tag: "distribution_version": "8.6"
    amazon-ebs.kib_image: Adding tag: "gpu_nvidia_version": ""
    amazon-ebs.kib_image: Adding tag: "kubernetes_cni_version": ""
    amazon-ebs.kib_image: Adding tag: "build_timestamp": "20231023182049"
    amazon-ebs.kib_image: Adding tag: "gpu_types": ""
    amazon-ebs.kib_image: Adding tag: "kubernetes_version": "1.28.7"
```

1231 https://cluster-api-aws.sigs.k8s.io/topics/images/built-amis.html
1232 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ami

```
==> amazon-ebs.kib_image: Creating snapshot tags
    amazon-ebs.kib_image: Adding tag: "ami_name": "konvoy-ami-
rhel-8.6-1.26.6-20231023182049"
==> amazon-ebs.kib_image: Terminating the source AWS instance...
==> amazon-ebs.kib_image: Cleaning up any extra volumes...
==> amazon-ebs.kib_image: No volumes to clean up, skipping
==> amazon-ebs.kib_image: Deleting temporary security group...
==> amazon-ebs.kib_image: Deleting temporary keypair...
==> amazon-ebs.kib_image: Running post-processor:  (type manifest)
Build 'amazon-ebs.kib_image' finished after 26 minutes 52 seconds.

==> Wait completed after 26 minutes 52 seconds

==> Builds finished. The artifacts of successful builds are:
--> amazon-ebs.kib_image: AMIs were created:
us-west-2: ami-04b8dfef8bd33a016

--> amazon-ebs.kib_image: AMIs were created:
us-west-2: ami-04b8dfef8bd33a016
```

After you complete these steps, you can Air-gapped Seed the Registry (see page 1611) .

If using AWS ECR as your local private registry, more information can be found on the Registry Mirror Tools (see page 1606) page.

9.3.3.5 Use Custom AMI to Build Cluster

9.3.3.5.1 Launch a DKP Cluster with a Custom AMI

To use the built `ami` with DKP, specify it with the `--ami` flag when calling cluster create. By default `konvoy-image` will name the AMI in such a way that `dkp` can discover the latest AMI for a base OS and Kubernetes version.

```
dkp create cluster aws --cluster-name=$(whoami)-aws-cluster --ami ami-0123456789
```

OR

To launch a DKP Cluster with Custom AMI **Lookup** that will use the latest AMI, specify the `--ami-format`, `--ami-base-os` and `--ami-owner` flags, use this command instead:

```
dkp create cluster aws --cluster-name=$(whoami)-aws-cluster --ami-format "konvoy-ami-
{{.BaseOS}}-?{{.K8sVersion}}-*" --ami-base-os centos-7 --ami-owner 123456789012
```

9.3.3.5.1.1 Using Custom Source AMIs

When using KIB for building machine images to Amazon, the default source AMIs that we provide are modeled by looking up an AMI based on the owner. Then we apply a filter for that operating system and version. At times, a particular upstream AMI may not be available in your region or something could be renamed so you create a custom source AMI.

You can view an example of that with the provided `centos-79.yaml` snippet below:

```
download_images: true

packer:
  ami_filter_name: "CentOS Linux 7"
  ami_filter_owners: "125523088429"
  distribution: "CentOS"
  distribution_version: "7.9"
  source_ami: ""
  ssh_username: "centos"
  root_device_name: "/dev/sda1"
...
```

Other times you want to provide a custom AMI. If this is the case, you will want to edit or create your own YAML file that looks up based on the `source_ami` field. For example, you can select images that are otherwise deprecated.

Once you select the source AMI that you want, you can declare that when running your build command:

```
konvoy-image build aws path/to/ami/centos-79.yaml --source-ami ami-0123456789
```

Alternatively, if you want to add it to your YAML file, or make your own file, you can do that as well. You add that AMI ID into the `source_ami` in the YAML file:

```
download_images: true

packer:
  ami_filter_name: ""
  ami_filter_owners: ""
  distribution: "CentOS"
  distribution_version: "7.9"
  source_ami: "ami-123456789"
  ssh_username: "centos"
  root_device_name: "/dev/sda1"
...
```

When you're done selecting your `source_ami`, you can build your KIB image as you would normally:

```
konvoy-image build aws path/to/ami/centos-79.yaml
```

9.3.4 KIB for EKS

AWS EKS best practices discourage building custom images. The Amazon EKS Optimized AMI[1233] is the preferred way to deploy containers for EKS. If the image is customized, it breaks some of the autoscaling and security capabilities of EKS.

9.4 Using KIB with Azure

9.4.1 Learn how to build a custom Azure Image for use with DKP

This procedure describes how to use the Konvoy Image Builder (see page 1626) (KIB) to create a Cluster API[1234] compliant Azure Virtual Machine (VM) Image. The VM Image contains the base operating system you specify and all the necessary Kubernetes components. The Konvoy Image Builder uses variable `overrides` to specify the base image and container images to use in your new Azure VM image.

The default Azure image is not recommended for use in production. We suggest using KIB for Azure to build the image in order to take advantage of enhanced cluster operations. Explore the Customize your Image (see page 1661) topic for more options.
For more information regarding using the image in creating clusters, refer to the Azure Create a New Cluster (see page 1314) section of the documentation.

9.4.2 Prerequisites

Before you begin, you must:

- Download the Konvoy Image Builder (see page 1626) bundle for your version of DKP.
- Check the Supported Infrastructure Operating Systems (see page 67)
- Check the Supported Kubernetes Version (see page 1706) for your Provider.
- Create a working `Docker` setup.

1233 https://docs.aws.amazon.com/eks/latest/userguide/eks-optimized-amis.html
1234 https://cluster-api.sigs.k8s.io/

9.4.3 Extract the KIB Bundle

Extract the bundle and `cd` into the extracted `konvoy-image-bundle-$VERSION_$OS` folder. The bundled version of `konvoy-image` contains an embedded `docker` image that contains all the requirements for building.

> The `konvoy-image` binary and all supporting folders are also extracted. When extracted, `konvoy-image` bind mounts the current working directory (`${PWD}`) into the container to be used.

9.4.4 Configure Azure Prerequisites

> If you have already followed the Azure Prerequisites (see page 1296) topic steps, then the environment variables needed by KIB (`[AZURE_CLIENT_SECRET` , `AZURE_CLIENT_ID` , `AZURE_TENANT_ID` , `AZURE_SUBSCRIPTION_ID`]) are set and do not need repeated if you are still working in the same window.

If you have not executed the Azure Prerequisite (see page 1296) steps, they are listed below.

1. Sign in to Azure:

```
az login
```


```
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "id": "b1234567-abcd-11a1-a0a0-1234a5678b90",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Mesosphere Developer Subscription",
    "state": "Enabled",
    "tenantId": "a1234567-b132-1234-1a11-1234a5678b90",
    "user": {
```

```
      "name": "user@azuremesosphere.onmicrosoft.com",
      "type": "user"
    }
  }
]
```

2. Create an Azure Service Principal (SP) by running the following command:

   If an SP with the name exists, this command will rotate the password.

```
az ad sp create-for-rbac --role contributor --name "$(whoami)-konvoy" --
scopes=/subscriptions/$(az account show --query id -o tsv) --query
"{ client_id: appId, client_secret: password, tenant_id: tenant }"
```

```
{
  "client_id": "7654321a-1a23-567b-b789-0987b6543a21",
  "client_secret": "Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C",
  "tenant_id": "a1234567-b132-1234-1a11-1234a5678b90"
}
```

3. Set the `AZURE_CLIENT_SECRET` environment variable:

```
export AZURE_CLIENT_SECRET="<azure_client_secret>" #
Z79yVstq_E.R0R7RUUck718vEHSuyhAB0C
export AZURE_CLIENT_ID="<client_id>"               # 7654321a-1a23-567b-
b789-0987b6543a21
export AZURE_TENANT_ID="<tenant_id>"               # a1234567-b132-1234-1a11-12
34a5678b90
export AZURE_SUBSCRIPTION_ID="<subscription_id>"   # b1234567-abcd-11a1-
a0a0-1234a5678b90
```

4. Ensure you have an override file (see page 1670) to configure specific attributes (see page 1678) of your Azure image.

Ensure you have named the correct file for your OS[1235] in the `konvoy-image build` command.

9.4.5 Build the Image

Run the `konvoy-image` command to build and validate the image.

1235 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/azure

```
konvoy-image build azure --client-id ${AZURE_CLIENT_ID} --tenant-id $
{AZURE_TENANT_ID} --overrides override-source-image.yaml images/azure/ubuntu-2004.yam
l
```

By default, the image builder builds in the `westus2` location. To specify another location set the `--location` flag (shown in example below is how to change the location to `eastus`):

```
konvoy-image build azure --client-id ${AZURE_CLIENT_ID} --tenant-id $
{AZURE_TENANT_ID} --location eastus --overrides override-source-image.yaml images/
azure/centos-7.yaml
```

When the command is complete, the image id is printed and written to the `./packer.pkr.hcl` file. This file has an `artifact_id` field whose value provides the name of the image. You should then specify this image id when creating the cluster.

9.4.6 Image Gallery

By default Konvoy Image Builder will create a Resource Group, Gallery, and Image Name to store the resulting image in. To specify a specific Resource Group, Gallery, or Image Name flags may be specified:

```
--gallery-image-locations string     a list of locations to publish the image
(default same as location)
--gallery-image-name string          the gallery image name to publish the image to
--gallery-image-offer string         the gallery image offer to set (default "dkp")
--gallery-image-publisher string     the gallery image publisher to set (default
"dkp")
--gallery-image-sku string           the gallery image sku to set
--gallery-name string                the gallery name to publish the image in
(default "dkp")
--resource-group string              the resource group to create the image in
(default "dkp")
```

When creating your cluster, you will then add this flag during the create process for your custom image: `--compute-gallery-id "<Managed Image Shared Image Gallery Id>"`. See Create a New Azure Cluster (see page 1314) for specific consumption of image commands.

The SKU and Image Name will default to the values found in the image YAML.

Ensure you have named the correct YAML file for your OS[1236] in the `konvoy-image build` command.

1236 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/gcp

9.4.7 Marketplace Images for Rocky Linux

Similar to Image Gallery, additional flags allow DKP to create a cluster with Marketplace based images for Rocky Linux 9.0: `--plan-offer`, `--plan-publisher` and `--plan-sku`.

- If you use these fields in the override file (see page 1670) when you create a machine image with KIB, you must also set the corresponding flags when you create your cluster with DKP.
- Conversely, if you do not use these fields when you create a machine image with KIB, you do not need to set these flags when you create your cluster with DKP.

Rocky Linux YAML

```
---
download_images: true

packer:
  distribution: "rockylinux-9"                # Offer
  distribution_version: "rockylinux-9"  # SKU
  # Azure Rocky linux official image: https://portal.azure.com/#view/
Microsoft_Azure_Marketplace/GalleryItemDetailsBladeNopdl/id/
erockyenterprisesoftwarefoundationinc1653071250513.rockylinux-9
  image_publisher: "erockyenterprisesoftwarefoundationinc1653071250513"
  image_version: "latest"
  ssh_username: "azureuser"
  plan_image_sku: "rockylinux-9" # SKU
  plan_image_offer: "rockylinux-9" # offer
  plan_image_publisher: "erockyenterprisesoftwarefoundationinc1653071250513" #
publisher

build_name: "rocky-90-az"
packer_builder_type: "azure"
python_path: ""
```

9.4.8 KIB for AKS

Because custom virtual machine images are discouraged in AKS, Konvoy Image Builder (KIB) does not include any support for building custom machine images for AKS. If the image is customized, it breaks some of the autoscaling and security capabilities of AKS. The AKS managed image ensures everything is patched and tuned for container workload.

9.5 Using KIB with GCP

This procedure describes how to use the Konvoy Image Builder (see page 1626) (KIB) to create a Cluster API[1237] compliant GCP image. GCP images contain configuration information and software to create a specific, pre-configured, operating environment. For example, you can create a GCP image of your current computer

1237 https://cluster-api.sigs.k8s.io/

system settings and software. The GCP image can then be replicated and distributed, creating your computer system for other users. KIB uses variable overrides (see page 1670) to specify base image and container images to use in your new GCP image.

Google Cloud Platform does not publish images. You must first build the image using Konvoy Image Builder. Explore the Customize your Image (see page 1661) topic for more options. For more information regarding using the image in creating clusters, refer to the GCP Infrastructure (see page 1486) section of the documentation.

9.5.1 DKP Prerequisites

Before you begin, you must:

- Download the KIB (see page 0) bundle for your version of DKP.
- Check the Supported Infrastructure Operating Systems (see page 67)
- Check the Supported Kubernetes Version (see page 1706) for your Provider.
- Create a working Docker or other registry setup.
- On Debian-based Linux distributions, install a version of the cri-tools[1238] package known to be compatible with both the Kubernetes and container runtime versions.
- Verify that your Google Cloud project does not have the Enable OS Login feature enabled. See below for more information:

The Enable OS Login feature is sometimes enabled by default in GCP projects. If the OS login feature is enabled, KIB will not be able to `ssh` to the VM instances it creates and will not be able to successfully create an image.
To check if it is enabled, use the commands on this page https://cloud.google.com/compute/docs/metadata/setting-custom-metadata#console_2 to inspect the metadata configured in in your project. If you find the the `enable-oslogin` flag set to TRUE, you must remove (or set it to FALSE) to successfully use KIB.

9.5.2 GCP Prerequisite Roles

If you are creating your image on either a non-GCP instance or one that does not have the required roles (see page 1488), you must either:

- Create a GCP service account[1239].
- If you have already created a service account, retrieve the credentials for an existing service account.

1238 https://github.com/kubernetes-sigs/cri-tools
1239 https://cloud.google.com/iam/docs/service-account-overview

- Export the static credentials that will be used to create the cluster:

```
export GCP_B64ENCODED_CREDENTIALS=$(base64 < "$
{GOOGLE_APPLICATION_CREDENTIALS}" | tr -d '\n')
```

Make sure to rotate static credentials for increased security.

9.5.3 Build the GCP image

1. Run the `konvoy-image` command to build and validate the image:

```
./konvoy-image build gcp --project-id ${GCP_PROJECT} --network ${NETWORK_NAME}
images/gcp/ubuntu-2004.yaml
```

2. KIB will run and print out the name of the created image, you will use this name when creating a Kubernetes cluster. See sample output below:

```
...
==> ubuntu-2004-focal-v20220419: Deleting instance...
    ubuntu-2004-focal-v20220419: Instance has been deleted!
==> ubuntu-2004-focal-v20220419: Creating image...
==> ubuntu-2004-focal-v20220419: Deleting disk...
    ubuntu-2004-focal-v20220419: Disk has been deleted!
==> ubuntu-2004-focal-v20220419: Running post-processor: manifest
Build 'ubuntu-2004-focal-v20220419' finished after 7 minutes 46 seconds.

==> Wait completed after 7 minutes 46 seconds

==> Builds finished. The artifacts of successful builds are:
--> ubuntu-2004-focal-v20220419: A disk image was created: konvoy-ubuntu-2004-1-2
3-7-1658523168
--> ubuntu-2004-focal-v20220419: A disk image was created: konvoy-ubuntu-2004-1-2
3-7-1658523168
```

Ensure you have named the correct YAML file for your OS[1240] in the `konvoy-image build` command.

1240 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/gcp

3. To find a list of images you have created in your account, run the following command:

```
gcloud compute images list --no-standard-images
```

Refer to DKP Documentation regarding roles and minimum permission for use with Konvoy Image Builder: GCP Roles (see page 1488)

9.5.4 Create a Network (optional)

Building an image requires a Network[1241] with firewall rules that allow SSH access to the VM instance.

1. Set your GCP Project ID for your `gcp` account unless already set previously:

```
export GCP_PROJECT=<your GCP project ID>
```

2. Run the following to create a new network:

```
export NETWORK_NAME=kib-ssh-network
gcloud compute networks create "${NETWORK_NAME}" --project="${GCP_PROJECT}" --
subnet-mode=auto --mtu=1460 --bgp-routing-mode=regional
```

3. Create the firewall rule to allow Ingress access on port 22:

```
gcloud compute firewall-rules create "${NETWORK_NAME}-allow-ssh" --project="$
{GCP_PROJECT}" --network="projects/${GCP_PROJECT}/global/networks/$
{NETWORK_NAME}" --description="Allows TCP connections from any source to any
instance on the network using port 22." --direction=INGRESS --priority=65534 --
source-ranges=0.0.0.0/0 --action=ALLOW --rules=tcp:22
```

With your KIB image now created, you can now move on to create your cluster and set up your Cluster API[1242] (CAPI) controllers.

9.6 Using KIB with GPU

Create a GPU supported OS image using Konvoy Image Builder

Using the Konvoy Image Builder (see page 1626), you can build an image that has support to use NVIDIA GPU hardware to support GPU workloads.

1241 https://cloud.google.com/vpc/docs/vpc
1242 https://cluster-api.sigs.k8s.io/

NOTE: The NVIDIA driver requires a specific Linux kernel version. Make sure that the base image for the OS version has the required kernel version.

See Supported Infrastructure Operating Systems (see page 67) for a list of OS versions and the corresponding kernel versions known to work with the NVIDIA driver.

If the NVIDIA runfile[1243] installer has not been downloaded, then retrieve and install the download first by running the following command. The first line in the command below downloads and installs the runfile and the second line places it in the artifacts directory (you must create an `artifacts` directory if you don't already have one).

```
curl -O https://download.nvidia.com/XFree86/Linux-x86_64/470.82.01/NVIDIA-Linux-
x86_64-470.82.01.run
mv NVIDIA-Linux-x86_64-470.82.01.run artifacts
```

DKP supported NVIDIA driver[1244] version is 470.x.

To build an image for use on GPU enabled hardware, perform the following steps.

1. In your override file, add the following to enable GPU builds. Otherwise, you can access and use the overrides in our repo[1245] or in the documentation under Nvidia GPU Override File (see page 1675) or Offline Nvidia Override file (see page 1676).
   a. Non-air-gapped GPU override `overrides/nvidia.yaml`:

      ```
      gpu:
        types:
          - nvidia
      build_name_extra: "-nvidia"
      ```

   b. Air-gapped GPU override `overrides/offline-nvidia.yaml`:

      ```
      # Use this file when building a machine image, not as a override secret
      for preprovisioned environments
      nvidia_runfile_local_file: "{{ playbook_dir}}/../artifacts/
      {{ nvidia_runfile_installer }}"
      gpu:
      ```

1243 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile
1244 https://www.nvidia.com/Download/Find.aspx
1245 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

```
  types:
    - nvidia

build_name_extra: "-nvidia"
```

NOTE: For RHEL Pre-provisioned Override Files (see page 1680) used with KIB, see specific note for GPU.

2. Build your image using the `build` Konvoy Image Builder command, making sure to include the flag `--instance-type` that specifies an AWS instance that has an available GPU:
   AWS Example:

   ```
   konvoy-image build --region us-east-1 --instance-type=p2.xlarge --source-
   ami=ami-12345abcdef images/ami/centos-7.yaml --overrides overrides/nvidia.yaml
   ```

   In this example, we chose an instance type with an NVIDIA GPU using the `--instance-type` flag, and we provided the NVIDIA overrides using the `--overrides` flag. See Using KIB with AWS (see page 1629) for more information on creating an AMI.

3. When the Konvoy Image Builder image is complete, the custom `ami` id is printed and written to `./.`. To use the built `ami` with Konvoy, specify it with the `--ami` flag when calling cluster create.

Additional helpful information can be found in the NVIDIA Device Plug-in[1246] for Kubernetes instructions and the Installation Guide of Supported Platforms[1247].

See also: NVIDIA documentation[1248]

9.6.1 Verification

To verify that the NVIDIA driver is working, connect to the node and execute this command:

```
nvidia-smi
```

When drivers are successfully installed, the display looks like the following:

```
Fri Jun 11 09:05:31 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.73.01    Driver Version: 460.73.01    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
```

1246 https://github.com/NVIDIA/k8s-device-plugin/blob/master/README.md
1247 https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html
1248 https://nvidia.custhelp.com/app/answers/detail/a_id/131/kw/driver%20installation%20docs/related/1

```
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:1E.0 Off |                    0 |
| N/A   35C    P0    73W / 149W |      0MiB / 11441MiB |     99%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

9.7 Using KIB with vSphere

vSphere for DKP using Konvoy Image Builder

This section assumes you have access to a vSphere environment and have credentials with proper privileges and access. Refer to the vCenter[1249] and vSphere Client[1250] documentation for details.

- Create a vSphere Base OS Image (see page 1652)
- Create a vSphere Virtual Machine Template (see page 1654)

9.7.1 Create a vSphere Base OS Image

Creating a base OS image from DVD ISO files is a one-time process. Building a base OS image creates a base vSphere template in your vSphere environment. The base OS image is used by Konvoy Image Builder (see page 1626)(KIB) to create a VM template to configure Kubernetes nodes by the DKP vSphere provider. For more information about images in vSphere vCenter, refer to VMware documentation[1251].

9.7.1.1 Create the Base OS Image

For vSphere, a username and password is populated by `SSH_USERNAME` and the user can use authorization via `SSH_PASSWORD` or `SSH_PRIVATE_KEY_FILE` environment variables and required by default by packer[1252]. This user should have administrator privileges. It is possible to configure a custom user and password when building the OS image, however, that requires the Konvoy Image Builder (KIB) configuration to be overridden (see page 1670).

While creating the base OS image, it is important to take into consideration the following elements:

1249 https://www.vmware.com/products/vcenter-server.html
1250 https://docs.vmware.com/en/VMware-vSphere/index.html
1251 https://docs.vmware.com/en/VMware-Cloud-Foundation/4.5/vcf-deploy/GUID-2674DA5A-8DF7-4212-A4A9-88CD798DC303.html
1252 https://github.com/mesosphere/konvoy-image-builder/blob/0523fd2c5e6e1ad1d4962f60a47039aa145a6e42/pkg/packer/manifests/vsphere/packer.pkr.hcl

- Storage configuration: D2iQ recommends customizing disk partitions and not configuring a SWAP partition.
- Network configuration: as KIB must download and install packages, activating the network is required.
- Connect to Red Hat: if using RHEL, registering with Red Hat is required to configure software repositories and install software packages.
- Software selection: D2iQ recommends choosing **Minimal Install**.
- DKP recommends to install with the packages provided by the operating system package managers. Use the version that corresponds to the major version of your operating system.

9.7.1.1.1 Disk Size

For each cluster you create using this base OS image, ensure you establish the disk size of the **root file system** based on:

- The minimum DKP Resource Requirements (see page 119).
- The minimum storage requirements for your organization.

9.7.1.1.1.1 Defaults

Clusters are created with a default disk size of 80 GB.

> ⚠ **For clusters created with the default disk size, the base OS image root file system must be exactly 80 GB.** The root file system cannot be reduced automatically when a machine first boots.

9.7.1.1.1.2 Customization

You can also specify a custom disk size when you create a cluster (see page 0) (see the flags available for use with the vSphere Create Cluster (see page 1861) command). This allows you to use one base OS image to create multiple clusters that have different storage requirements.

> ⚠ Before specifying a disk size when you create a cluster, take into account:
>
> 1. **For some base OS images, the custom disk size option has no effect on the size of the root file system.** This is because some root file systems, for example, those contained in an LVM Logical Volume, cannot be resized automatically when a machine first boots.
> 2. **The specified custom disk size must be equal to, or larger than the size of the base OS image root file system.** This is because a root file system cannot be reduced automatically when a machine first boots.

3. In VMware Cloud Director (see page 1444), the base image determines the minimum storage available for the VM.

For additional information and examples of KIB vSphere overrides, see this page: Base Image Override Files (see page 1678)

If using Flatcar, the documentation from Flatcar regarding disabling or enabling autologin in the Base OS Image is found here: In a vSphere or Pre-provisioned environment, anyone with access to the console of a Vitrual Machine(VM) has access to the core operating system user. This is called autologin. To disable autologin, you add paramaters to your base Flatcar image. The ways to do this are described in the Flatcar documentation: https://www.flatcar.org/docs/latest/installing/cloud/vmware/#disablingenabling-autologin and https://www.flatcar.org/docs/latest/setup/customization/other-settings/#adding-custom-kernel-boot-options. Refer to the vCenter[1253] and vSphere Client[1254] documentation for details.

9.7.1.1.2 Next Step

Create a vSphere Virtual Machine Template (see page 1654)

9.7.2 Create a vSphere Virtual Machine Template

Create a vSphere template for your cluster from a base OS image

You must have at least one image before creating a new cluster. As long as you have an image, this step in your configuration is not required each time since that image can be used to spin up a new cluster. However, if you need different images for different environments or providers, you will need to create a new custom image.

The Konvoy Image Builder (see page 1652) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. Using KIB, you can build an image without requiring access to the internet by providing an additional offline `--override` flag. You can use these overrides files to customize some of the components installed on your machine image. For example, you could tell KIB to install the FIPS versions of the Kubernetes components.

9.7.2.1 Prerequisites

- Users need to create a base OS image (see page 1652) in their vSphere client before starting this procedure.
- Konvoy Image Builder (KIB) downloaded (see page 76) and extracted
- **Air-gapped Only**
    - Assuming you have downloaded (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`, extract the tarball to a local directory:

1253 https://www.vmware.com/products/vcenter-server.html
1254 https://docs.vmware.com/en/VMware-vSphere/index.html

```
tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz && cd dkp-v2.8
.1/kib
```

- In previous DKP releases, the distro package bundles were included in the downloaded air-gapped bundle. Currently, that air-gapped bundle contains the following artifacts with the exception of the distro packages:
  - DKP Kubernetes packages
  - Python packages (provided by upstream)
  - Containerd tarball

  i. Download (see page 76) `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz` , and extract the tarball to a local directory:

  ```
  tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz && cd
  dkp-v2.8.1/kib
  ```

  ii. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.

  iii. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle` . This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml` . Example command:

  ```
  ./konvoy-image create-package-bundle --os redhat-8.4 --output-
  directory=artifacts
  ```

  NOTE: For FIPS, pass the flag: `--fips`

  NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY` . Example command:

  ```
  export RHSM_ACTIVATION_KEY="-ci"
  export RHSM_ORG_ID="1232131"
  ```

- Follow the instructions to build a vSphere template below and set the override `--overrides overrides/offline.yaml` flag.

9.7.2.2 Create a vSphere Template for Your Cluster from a Base OS Image

Using the base OS image created in a previous procedure, DKP creates the new vSphere template directly on the vCenter server.

1. Set the following vSphere environment variables on the bastion VM host:

```
export VSPHERE_SERVER=your_vCenter_APIserver_URL
export VSPHERE_USERNAME=your_vCenter_user_name
export VSPHERE_PASSWORD=your_vCenter_password
```

2. Copy the base OS image file created in the vSphere Client to your desired location on the bastion VM host and make a note of the path and file name.

3. Create an `image.yaml` file and add the following variables for vSphere. DKP uses this file and these variables as inputs in the next step. To customize your `image.yaml` file, refer to this section: Customize your Image (see page 1661).
⚠️ **NOTE**: This example is Ubuntu 20.04. You will need to replace OS name below based on your OS. See other default YAML examples[1255] for copy and paste below last step.

```
---
download_images: true
build_name: "ubuntu-2004"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-
init-vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: "<VSPHERE_CLUSTER_NAME>"
  datacenter: "<VSPHERE_DATACENTER_NAME>"
  datastore: "<VSPHERE_DATASTORE_NAME>"
  folder: "<VSPHERE_FOLDER>"
  insecure_connection: "false"
  network: "<VSPHERE_NETWORK>"
  resource_pool: "<VSPHERE_RESOURCE_POOL>"
  template: "os-qualification-templates/d2iq-base-Ubuntu-20.04" # change
default value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "ubuntu2004-64"
  # goss params
  distribution: "ubuntu"
  distribution_version: "20.04"
# Use following overrides to select the authentication method that can be used
with base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
```

1255 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

```
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key
will be ignored
```

4. Create a vSphere VM template with your variation of the following command:

   ```
   konvoy-image build images/ova/<image.yaml>
   ```

   Any additional configurations can be added to this command using `--overrides` flags as shown below:

   a. Any credential overrides: `--overrides overrides.yaml`
   b. for FIPS, add this flag: `--overrides overrides/fips.yaml`
   c. for air-gapped, add this flag: `--overrides overrides/offline-fips.yaml`

5. The Konvoy Image Builder (see page 1652) (KIB) uses the values in `image.yaml` and the input base OS image to create a vSphere template directly on the vCenter server. This template contains the required artifacts needed to create a Kubernetes cluster.
   When KIB provisions the OS image[1256] successfully, it creates a manifest file. The `artifact_id` field of this file contains the name of the AMI ID (AWS), template name (vSphere), or image name (GCP/Azure), for example:

   ```
   {
         "name": "vsphere-clone",
         "builder_type": "vsphere-clone",
         "build_time": 1644985039,
         "files": null,
         "artifact_id": "konvoy-ova-vsphere-rhel-84-1.21.6-1644983717",
         "packer_run_uuid": "260e8110-77f8-ca94-e29e-ac7a2ae779c8",
         "custom_data": {
           "build_date": "2022-02-16T03:55:17Z",
           "build_name": "vsphere-rhel-84",
           "build_timestamp": "1644983717",
           [...]
         }
       }
   ```

   Recommendation: Now we can now see the template created in our vCenter, it is best to rename it to `dkp-<DKP_VERSION>-k8s-<K8S_VERSION>-<DISTRO>`, like `dkp-2.4.0-k8s-1.24.6-ubuntu` to keep templates organized.

6. Next steps are to deploy a DKP cluster using your vSphere template.

1256 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

9.7.2.2.1 Additional OS YAML files for copy/paste below or can be found in GitHub[1257]:

RHEL 8.6

```
---
download_images: true
build_name: "rhel-86"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "os-qualification-templates/d2iq-base-RHEL-86" # change default value
with your base template name
  vsphere_guest_os_type: "rhel8_64Guest"
  guest_os_type: "rhel8-64"
  # goss params
  distribution: "RHEL"
  distribution_version: "8.6"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Rocky Linux 9.1

```
---
download_images: true
build_name: "rocky-91"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
```

1257 https://github.com/mesosphere/konvoy-image-builder/tree/main/images/ova

```
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "os-qualification-templates/d2iq-base-RockyLinux-9.1" # change default
 value with your base template name
  vsphere_guest_os_type: "other4xLinux64Guest"
  guest_os_type: "rocky9-64"
  # goss params
  distribution: "rocky"
  distribution_version: "9.1"
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: false  # is set to true, ssh_password and ssh_private_key will be
ignored
```

Flatcar

```
---
download_images: true
build_name: "flatcar-3033.3.16"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: ""
  datacenter: ""
  datastore: ""
  folder: ""
  insecure_connection: "false"
  network: ""
  resource_pool: ""
  template: "d2iq-base-templates/d2iq-base-Flatcar-3033.3.16"
  vsphere_guest_os_type: "flatcar_64Guest"
  guest_os_type: "flatcar-64"
  # goss params
  distribution: "flatcar"
  distribution_version: "3033.3.16"
```

```
# Use following overrides to select the authentication method that can be used with
base template
# ssh_username: ""  # can be exported as environment variable 'SSH_USERNAME'
# ssh_password: "" # can be exported as environment variable 'SSH_PASSWORD'
# ssh_private_key_file = "" # can be exported as environment variable
'SSH_PRIVATE_KEY_FILE'
# ssh_agent_auth: true  # is set to true, ssh_password and ssh_private_key will be
ignored
```

9.8 Using KIB with Pre-provisioned Environments

In Pre-provisioned environments, KIB runs automatically inside the `bootstrap cluster`, against the list of nodes that you define. Whereas with other providers, you run KIB outside the cluster manually to build your images.

For Pre-provisioned, you define a set of nodes that already exist. During the cluster creation process, KIB is built into DKP and automatically runs the machine configuration process (which KIB uses to build images for other providers) against the set of nodes that you defined. This results in your pre-existing or **pre-provisioned** nodes being configured properly. The remainder of the cluster provisioning happens automatically after that.

- In a **Pre-provisioned environment** (see page 148), you have existing machines and DKP consumes them to form a cluster.
- When you have another provisioner (for example, cloud providers such AWS (see page 294), vSphere (see page 441) and others), you build images with KIB and DKP consumes the images to provision machines and form a cluster.

Prior to DKP 2.6, you had to specify the HTTP proxy in the KIB override setup and then again in the `dkp create cluster` command. After DKP 2.6, an HTTP proxy gets created from the Konvoy flags for the control plane proxy and workers proxy values. The flags in the DKP command for Pre-provisioned clusters populate a Secret automatically in the bootstrap cluster. That Secret has a known name that the Pre-provisioned controller finds and applies when it runs the KIB provisioning job.

For a Pre-provisioned **air-gapped environment**, you will have to build the OS packages after fetching the packages from the distro repositories.

In previous DKP releases, the distro package bundles were included in the downloaded air-gapped bundle. Currently, that air-gapped bundle contains the following artifacts with the exception of the distro packages:

- DKP Kubernetes packages
- Python packages (provided by upstream)
- Containerd tarball

1. Download (see page 76) `dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz`, and extract the tarball to a local directory:

```
tar -xzvf dkp-air-gapped-bundle_v2.8.0_linux_amd64.tar.gz && cd dkp-v2.8.0/kib
```

2. You will need to fetch the distro packages as well as other artifacts. By fetching the distro packages from distro repositories, you get the latest security fixes available at machine image build time.
3. In your download location, there is a bundles directory with all the steps to create an OS package bundle for a particular OS. To create it, run the new DKP command `create-package-bundle`. This builds an OS bundle using the Kubernetes version defined in `ansible/group_vars/all/defaults.yaml`. Example command:

```
./konvoy-image create-package-bundle --os redhat-8.4 --output-
directory=artifacts
```

NOTE: For FIPS, pass the flag: `--fips`

NOTE: For RHEL OS, pass your RedHat subscription manager credentials: `export RMS_ACTIVATION_KEY`. Example command:

```
export RHSM_ACTIVATION_KEY="-ci"
export RHSM_ORG_ID="1232131"
```

9.9 Customize your Image

There are several ways in which to customize an image. Some methods are simple and some are more complex customizations. Each section in this portion of the documentation will explain these options and give directions.

- Customize your Image YAML or Manifest File (see page 1661)
- Customize Packer Configuration (see page 1665)
- Add Custom Tags to your Image (see page 1668)
- Ansible Variables (see page 1669)
- Using Override Files with Konvoy Image Builder (see page 1670)
    - Default Override Files (see page 1671)
    - Custom Override Files (see page 1677)

9.9.1 Customize your Image YAML or Manifest File

DKP provides default YAML files for each Infrastructure Provider and their supported Operating Systems. Those default YAML files can easily be modified by replacing values. The simplest way is to open the default

YAML[1258] for your OS in an editor and change a line. For example, in the following YAML, you can name a specific image rather than accepting the default `source_ami: ""`

AWS Example: `# Example images/ami/rhel-86.yaml`

```
---
# Example images/ami/rhel-86.yaml
download_images: true

packer:
  ami_filter_name: "RHEL-8.6.0_HVM-*"
  ami_filter_owners: "309956199498"
  distribution: "RHEL"
  distribution_version: "8.6"
  source_ami: ""
  ssh_username: "ec2-user"
  root_device_name: "/dev/sda1"
  volume_size: "15"

build_name: "rhel-8.6"
packer_builder_type: "amazon"
python_path: ""
```

Once the YAML for the image is edited, you then create your image using the customized YAML by applying it in the command.

```
konvoy-image build aws images/ami/rhel-86.yaml
```

9.9.1.1 Generate a Packer File to Customize

Another method for customization is to edit your Packer template. When first running the Konvoy Image Builder(KIB) command to generate an image, it will automatically create the files needed for you to edit in a path called work folder.

First you will build an image and then you will edit the generated output to use in customizing another image. During the attempt to build the image, a `packer.pkr.hcl` is generated under the path `work/centos-7-xxxxxxx-yyyyy/packer.pkr.hcl`.

1. Run the image creation command:

   ```
   konvoy-image build aws images/ami/ubuntu-2004.yaml
   ```

Overrides for FIPS

1258 https://github.com/mesosphere/konvoy-image-builder/tree/7447074a6d910e71ad2e61fc4a12820d073ae5ae/images

You can also use FIPS overrides during initial image creation:

```
konvoy-image build aws images/ami/ubuntu-2004.yaml --overrides overrides/fips.yaml
```

To avoid fully creating the image and reduce image charges, interrupt the command right after running it. This will still generate the needed files but avoid a full image creation charge.

2. A *work* file is generated (EX: `work/ubuntu-20-3486702485-iulvY`) which contains the files to be opened in an editor and the values added or changed:
    a. `ansible_vars.yaml`
    b. `packer_vars.json`
    c. `packer.pkr.hcl`
3. Open the `packer.pkr.hcl` file with an editor and change the values desired.

packer.pkr.hcl

~/Development/d2iq/kib-the-app/konvoy-image-bundle-v2.8.1_darwin_arm64/work/rhel-8.4-fips

```
packer {
  required_plugins {
    amazon = {
      version = ">= 1.1.3"
      source = "github.com/hashicorp/amazon"
    }
    ansible = {
      version = ">= 1.1.0"
      source  = "github.com/hashicorp/ansible"
    }
    goss = {
      version = ">=3.1.5"
      source = "github.com/supershal/goss"
    }
  }
}

variable "ami_groups" {
  type    = string
  default = ""
}

variable "ami_regions" {
  type    = string
  default = "us-west-2"
}

variable "ami_users" {
  type    = string
  default = ""
}

variable "ansible_extra_vars" {
  type    = string
  default = ""
}

variable "aws_access_key" {
  type    = string
  default = ""
}

variable "aws_instance_type" {
  type    = string
  default = "t3.small"
}

variable "aws_profile" {
  type    = string
  default = ""
}

variable "aws_region" {
  type    = string
  default = "us-west-2"
}
```

4. Save and rename these changes as `manifest.pkr.kcl` so that it can be applied during image creation using the `--packer-manifest` flag.

For a full list of KIB flags, run the command: `konvoy-image build aws --help`

5. Once the YAML for the image is edited, you then create another image using your customized YAML by applying it with the flag `--packer-manifest`. Provision the new image applying the recently edited and renamed `manifest.pkr.kcl`:

   AWS example:

   ```
   konvoy-image build aws --packer-manifest manifest.pkr.hcl --overrides
   overrides/fips.yaml
   ```

6. Check final image for correct variables applied.

9.9.1.1.1 Next Topic

Customize Packer Configuration (see page 1665)

9.9.2 Customize Packer Configuration

Although there are several parameters specified by default in the Packer templates for each provider, it is possible to **override** the default values.

For a complete list of the variables that can be modified for each Packer builder, users can refer to:

- AWS Packer template[1259]
- Azure Packer template[1260]
- GCP Packer template[1261]
- vSphere Packer template[1262]

9.9.2.1 Using Flags

One option is to execute KIB with specific flags to override the following values:

1. the source AMI (`--source-ami`)
2. AMI region (`--ami-regions`)
3. AWS EC2 instance type (`--aws-instance-type`)

For a comprehensive list of these flags, please run:

```
./konvoy-image build --help
```

1259 https://github.com/mesosphere/konvoy-image-builder/blob/main/pkg/packer/manifests/aws/packer.pkr.hcl
1260 https://github.com/mesosphere/konvoy-image-builder/blob/main/pkg/packer/manifests/azure/packer.pkr.hcl
1261 https://github.com/mesosphere/konvoy-image-builder/blob/main/pkg/packer/manifests/gcp/packer.pkr.hcl
1262 https://github.com/mesosphere/konvoy-image-builder/blob/main/pkg/packer/manifests/vsphere/packer.pkr.hcl

9.9.2.2 Using Override Files

Another option is by creating a file with the parameters to be overridden and specify the `--overrides` flag as shown below:

```
./konvoy-image build images/<builder-type>/<os>-<version>.yaml --overrides
overrides.yaml
```

> While CLI flags can be used in combination with override files, CLI flags take priority over any override files.

DKP provides several default override files and explains how to create your own custom override files as well. Refer to whichever information you need:

- Default Override Files (see page 1671)
- Custom Override Files (see page 1677)

9.9.2.3 Override Credentials in Packer

To abide to security practices, a user could set their own username and password while creating the base OS image and override the default credentials in KIB.

AWS Example:

For example, when using the AWS Packer builder to override the credentials, set them under the `packer` key. This overrides the image search and forces the use of the specified credentials.

Create your override file `overrides-packer-credentials.yaml`:

```
---
packer:
  ssh_username: "<USERNAME>"
  ssh_password: "<PASSWORD>"
```

After creating the override file for your credentials, pass your override file by using the `--overrides` flag when building our image:

```
./konvoy-image build aws images/ami/centos-7.yaml --overrides override-packer-
credentials.yaml
```

Azure Example:

An Azure example would be the current base image description at `images/azure/centos-79.yaml` which is similar to the following:

```
---
# Example images/azure/centos-79.yaml
download_images: true
packer:
  distribution: "centos"  #offer
  distribution_version: "7_9-gen2"  #SKU
  image_publisher: "openlogic"
  image_version: "latest"
  ssh_username: "centos"

build_name: "centos-7"
packer_builder_type: "azure"
python_path: ""
```

or

vSphere Example:

A vSphere example would be the current base image description at `images/vsphere/rhel-79.yaml` which is similar to the following:

```
---
download_images: true
build_name: "rhel-79"
packer_builder_type: "vsphere"
guestinfo_datasource_slug: "https://raw.githubusercontent.com/vmware/cloud-init-
vmware-guestinfo"
guestinfo_datasource_ref: "v1.4.0"
guestinfo_datasource_script: "{{guestinfo_datasource_slug}}/
{{guestinfo_datasource_ref}}/install.sh"
packer:
  cluster: "zone1"
  datacenter: "dc1"
  datastore: "esxi-06-disk1"
  folder: "cluster-api"
  insecure_connection: "false"
  network: "Airgapped"
  resource_pool: "Users"
  template: "base-rhel-7"
  vsphere_guest_os_type: "rhel7_64Guest"
  guest_os_type: "rhel7-64"
  #goss params
  distribution: "RHEL"
  distribution_version: "7.9"
```

See Supported Operating Systems (see page 67) for details.

9.9.2.3.1 Next Topic

Add Custom Tags to your Image (see page 1668)

9.9.3 Add Custom Tags to your Image

Some cloud environments enforce the tagging of resources. Is it possible to add custom tags on an image that was built with Konvoy Image Builder(KIB)? Currently, adding a custom tag as an *attribute* to the image is only possible by modifying the `packer.pkr.hcl` file.

In order to add a tag, perform the commands below:

1. Generate manifest files for KIB by executing the following command. In this AWS example, we're building a CentOS 7.9 image:

   ```
   ./konvoy-image build images/ami/centos-79.yaml
   ```

> Other provider commands are located in the Konvoy Image Builder (see page 1626) under each provider.

2. Send `SIGINT` to the process to kill it after seeing the output `...writing new packer configuration to work/centos-7-xxxxxxx-yyyyy`. During the attempt to build the image, a `packer.pkr.hcl` is generated under the path `work/centos-7-xxxxxxx-yyyyy/packer.pkr.hcl`.
3. Edit the `packer.pkr.hcl` file by adding the parameter `"run_tags"` to the `packer.pkr.hcl` file as seen in the AWS example below:

   ```
    "builders": [
   {
   "name": "{{(user `distribution`) | lower}}-{{user `distribution_version`}}
   {{user `build_name_extra`}}",
   "type": "amazon-ebs",
   "run_tags": {"my_custom_tag": "tag"},
   "instance_type": "{{user `aws_instance_type`}}",
   "source_ami_filter": {
   "filters": {
   "virtualization-type": "hvm",
   "name": "{{user `ami_filter_name`}}",
   "root-device-type": "ebs",
   "architecture": "x86_64"
   },
   ```

4. Use the `--packer-manifest` flag to apply it when you build the image:

```
./konvoy-image build aws images/ami/centos-79.yaml --packer-manifest=work/
centos-7-1658174984-TycMM/packer.pkr.hcl
2022/09/07 18:23:33 writing new packer configuration to work/centos-7-166257501
3-zJUhP
2022/09/07 18:23:33 starting packer build
centos-7.9: output will be in this color.

==> centos-7.9: Prevalidating any provided VPC information
==> centos-7.9: Prevalidating AMI Name: konvoy-ami-centos-7-1.26.3-1662575013
centos-7.9: Found Image ID: ami-0686851c4e7b1a8e1
==> centos-7.9: Creating temporary keypair: packer_6318e1a6-9f45-c01b-c7ca-
f5404735f709
==> centos-7.9: Creating temporary security group for this instance:
packer_6318e1aa-7448-d74b-9b90-166f26d21619
==> centos-7.9: Authorizing access to port 22 from [0.0.0.0/0] in the temporary
security groups...
==> centos-7.9: Launching a source AWS instance...
centos-7.9: Adding tag: "my_custom_tag": "tag"
centos-7.9: Instance ID: i-04d457b20713dcf7d
==> centos-7.9: Waiting for instance (i-04d457b20713dcf7d) to become ready...
==> centos-7.9: Using SSH communicator to connect: 34.222.153.107
==> centos-7.9: Waiting for SSH to become available...
```

Now that your tags have been added to the image, you can continue to Custom Installation and Additional Infrastructure Tools (see page 1050). There you will build the image with your edited manifest and finish building your clusters under your infrastructure.

9.9.3.1 Next Topic

Using Override Files with Konvoy Image Builder (see page 1670)

9.9.4 Ansible Variables

Ansible variables are generally preset when using KIB. To change the variables, you change the Ansible runbook or preprogrammed playbook.

Example `ansible_vars.yaml`:

```
build_name: ubuntu-20
download_images: true
kubernetes_full_version: 1.28.7
kubernetes_version: 1.28.7
```

```
packer:
  ami_filter_name: ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server*
  ami_filter_owners: "099720109477"
  distribution: Ubuntu
  distribution_version: "20.04"
  dry_run: false
  goss_arch: amd64
  goss_entry_file: goss/goss.yaml
  goss_format: json
  goss_format_options: pretty
  goss_inspect_mode: false
  goss_tests_dir: goss
  goss_url: null
  goss_vars_file: ansible/group_vars/all/system.yaml
  goss_version: 0.3.16
  root_device_name: /dev/sda1
  source_ami: ""
  ssh_username: ubuntu
  volume_size: "15"
packer_builder_type: amazon
python_path: ""
```

9.9.5 Using Override Files with Konvoy Image Builder

9.9.5.1 Learn how to use override files with Konvoy Imgage builder

You can use overrides files to customize some of the components installed on your machine image. For example, you could tell KIB to install the FIPS versions of the Kubernetes components. You could also add `offline` overrides used for air-gapped environments. That will upload the necessary OS packages and other files to the image, so that KIB doesn't try to look for those using the Internet. Or you could use an override file to provide similar configurations for using a GPU-enabled cluster. The KIB base override files are located in this Github repository[1263].

You can only change certain, predefined parameters using this mechanism. The set of parameters you can change depends on which of the supported infrastructure providers you are using. For example, for vSphere, there are many settings that you use to tell the KIB how to talk to your vCenter implementation, where to store the images, which credentials to use, and so on.

Konvoy Image Builder (see page 1626) uses `override` files to configure specific attributes. These files provide information to override default values for certain parts of your image file. These `override` files can modify the version and parameters of the image description and the Ansible playbook.

There are 2 types of override files:

- Default override (see page 1671) files provided by Konvoy Image Builder for you to use for specific purposes.
- Custom override files (see page 1677) you create to use with Konvoy Image Builder.

1263 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

9.9.5.2 Default Override Files

9.9.5.2.1 Learn how to use override files with DKP

DKP comes with default override files:

- FIPS Override Non-air-gapped Files (see page 1671)
- FIPS Override Air-gapped Environment Files (see page 1673)
- Offline Override File (see page 1674)
- Nvidia GPU Override File (see page 1675)
- Offline Nvidia GPU Override file (see page 1676)
- Oracle Redhat Linux Kernel Override File (see page 1677)

You can find these override files in the Konvoy Image Builder repo[1264].

9.9.5.2.2 Related Topic

Custom Override Files (see page 1677)

9.9.5.2.3 FIPS Override Non-air-gapped Files

9.9.5.2.3.1 Cloud Provisioners Override File:

Online FIPS Override File (Non-air-gapped)

Add the following FIPS override file to your environment:

`--overrides overrides/fips.yaml`

```
---
k8s_image_registry: docker.io/mesosphere

fips:
  enabled: true

build_name_extra: -fips
kubernetes_build_metadata: fips.0
default_image_repo: hub.docker.io/mesosphere
kubernetes_rpm_repository_url: "https://packages.d2iq.com/konvoy/stable/linux/repos/
el/kubernetes-v{{ kubernetes_version }}-fips/x86_64"
docker_rpm_repository_url: "\
  https://containerd-fips.s3.us-east-2.amazonaws.com\
```

1264 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

```
/{{ ansible_distribution_major_version|int }}\
/x86_64"
```

You can find all available Overrides files in the Konvoy Image Builder repo[1265].

9.9.5.2.3.2 Pre-provisioned Environments Override File:

Online FIPS Override File (Pre-provisioned)

Add the following FIPS override file to your environment:

1. If your pre-provisioned machines need to have a default Override file like FIPS, create a secret that includes the overrides in a file:

```
cat > fips.yaml << EOF
---
k8s_image_registry: docker.io/mesosphere

fips:
  enabled: true

build_name_extra: -fips
kubernetes_build_metadata: fips.0
default_image_repo: hub.docker.io/mesosphere
kubernetes_rpm_repository_url: "https://packages.d2iq.com/konvoy/stable/linux/
repos/el/kubernetes-v{{ kubernetes_version }}-fips/x86_64"
docker_rpm_repository_url: "\
  https://containerd-fips.s3.us-east-2.amazonaws.com\
  /{{ ansible_distribution_major_version|int }}\
  /x86_64"
EOF
```

2. Create the related secret by running the following command:

```
kubectl create secret generic $CLUSTER_NAME-user-overrides --from-
file=fips.yaml=fips.yaml
kubectl label secret $CLUSTER_NAME-user-overrides clusterctl.cluster.x-k8s.io/
move=
```

1265 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

You can find all available Overrides files in the Konvoy Image Builder repo[1266].

9.9.5.2.4 FIPS Override Air-gapped Environment Files

9.9.5.2.4.1 Cloud Provisioners Override File:

Offline FIPS Override File (Air-gapped)

Add the following FIPS offline override file to your environment:

`--overrides overrides/offline-fips.yaml`

```
# fips os-packages
os_packages_local_bundle_file: "{{ playbook_dir }}/../artifacts/
{{ kubernetes_version }}_{{ ansible_distribution|lower }}
_{{ ansible_distribution_major_version }}_x86_64_fips.tar.gz"
containerd_local_bundle_file: "{{ playbook_dir }}/../artifacts/
{{ containerd_tar_file }}"
pip_packages_local_bundle_file: "{{ playbook_dir }}/../artifacts/pip-packages.tar.gz"
images_local_bundle_dir: "{{ playbook_dir}}/../artifacts/images"
```

You can find all available Overrides files in the Konvoy Image Builder repo[1267].

9.9.5.2.4.2 Pre-provisioned Environments Override File:

Offline FIPS Override File (Air-gapped)

Add the following FIPS offline override file to your environment:

1. If your pre-provisioned machines need to have a default Override file like FIPS, create a secret that includes the overrides in a file:

1266 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides
1267 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

```
cat > fips.yaml << EOF
# fips os-packages
os_packages_local_bundle_file: "{{ playbook_dir }}/../artifacts/
{{ kubernetes_version }}_{{ ansible_distribution|lower }}
_{{ ansible_distribution_major_version }}_x86_64_fips.tar.gz"
containerd_local_bundle_file: "{{ playbook_dir }}/../artifacts/
{{ containerd_tar_file }}"
pip_packages_local_bundle_file: "{{ playbook_dir }}/../artifacts/pip-
packages.tar.gz"
images_local_bundle_dir: "{{ playbook_dir}}/../artifacts/images"
EOF
```

2. Create the related secret by running the following command:

```
kubectl create secret generic $CLUSTER_NAME-user-overrides --from-
file=fips.yaml=fips.yaml
kubectl label secret $CLUSTER_NAME-user-overrides clusterctl.cluster.x-k8s.io/
move=
```

You can find all available Overrides files in the Konvoy Image Builder repo[1268].

9.9.5.2.4.3 Related Topic

Private Registry in Air-gapped Override (see page 1683)

9.9.5.2.5 Offline Override File

You can find these offline (air-gapped) override files in the Konvoy Image Builder repo[1269].

`--overrides overrides/offline.yaml`

```
os_packages_local_bundle_file: "{{ playbook_dir }}/../artifacts/
{{ kubernetes_version }}_{{ ansible_distribution|lower }}
_{{ ansible_distribution_major_version }}_x86_64.tar.gz"
containerd_local_bundle_file: "{{ playbook_dir }}/../artifacts/
{{ containerd_tar_file }}"
pip_packages_local_bundle_file: "{{ playbook_dir }}/../artifacts/pip-packages.tar.gz"
images_local_bundle_dir: "{{ playbook_dir}}/../artifacts/images"
```

1268 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides
1269 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

For Ubuntu 20.04, when Konvoy Image Builder runs, it will temporarily disable all defined Debian repositories by appending a `.disabled` suffix. At the end of installation, each repository will revert to its original name. In case of failures, the files will not be renamed back.

For GPU Offline Override File (see page 1676), you can use `nvidia-runfile` flag for GPU support if you have downloaded the runfile installer[1270].

9.9.5.2.5.1 Related Topics

Offline Nvidia GPU Override file (see page 1676)

Private Registry in Air-gapped Override (see page 1683)

9.9.5.2.5.2 Next Topic

Nvidia GPU Override File (see page 1675)

9.9.5.2.6 Nvidia GPU Override File

9.9.5.2.6.1 GPU Override File

This override file should be used to create images for use with DKP that can use GPU hardware. These override files can also be located in the Konvoy Image Builder repo[1271].

```
--overrides overrides/nvidia.yaml
```

```
---
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
```

9.9.5.2.6.2 GPU Override File for Pre-provisioned environments

If you require your environment to consume GPU resources, add the following FIPS Overrides file to your environment:

1270 https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#runfile
1271 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

1. If your pre-provisioned machines need to have a default Override file like FIPS, create a secret that includes the overrides in a file:

```
cat > nvidia.yaml << EOF
---
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
EOF
```

2. Create the related secret by running the following command:

```
kubectl create secret generic $CLUSTER_NAME-user-overrides --from-
file=nvidia.yaml=nvidia.yaml
kubectl label secret $CLUSTER_NAME-user-overrides clusterctl.cluster.x-k8s.io/
move=
```

9.9.5.2.7 Offline Nvidia GPU Override file

This override file should be used to create images for use with DKP that can use GPU hardware. These override files can also be located in the Konvoy Image Builder repo[1272].
`--overrides overrides/offline-nvidia.yaml`

```
# Use this file when building a machine image, not as a override secret for
 preprovisioned environments
nvidia_runfile_local_file: "{{ playbook_dir}}/../artifacts/
{{ nvidia_runfile_installer }}"
gpu:
  types:
    - nvidia

build_name_extra: "-nvidia"
```

9.9.5.2.7.1 GPU Override File for Air-gapped Pre-provisioned Environments

If you require your environment to consume GPU resources, add the following GPU Overrides file to your environment:

1. If your pre-provisioned machines need to have a default Override file like GPU, create a secret that includes the overrides in a file:

1272 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

```
cat > nvidia.yaml << EOF
---
nvidia_runfile_local_file: "{{ playbook_dir}}/../artifacts/
{{ nvidia_runfile_installer }}"
gpu:
  types:
    - nvidia
build_name_extra: "-nvidia"
EOF
```

2. Create the related secret by running the following command:

```
kubectl create secret generic $CLUSTER_NAME-user-overrides --from-
file=nvidia.yaml=nvidia.yaml
kubectl label secret $CLUSTER_NAME-user-overrides clusterctl.cluster.x-k8s.io/
move=
```

9.9.5.2.8 Oracle Redhat Linux Kernel Override File

You can find these override files in the Konvoy Image Builder repo[1273].

`--overrides overrides/rhck.yaml`

```
---
oracle_kernel: RHCK
```

9.9.5.3 Custom Override Files

You can customize your KIB images using override files to specify alternate package libraries, Docker image repos, HTTP/S proxy information, and other customizations.

Some example custom override files are described in the following topics:

- Image Overrides (see page 1678)
- Pre-Provisioned Override Files (see page 1680)
- Using HTTP/S Proxy with KIB Images (see page 1681)
- Private Registry in Air-gapped Override (see page 1683)

9.9.5.3.1 Related Topic

Default Override Files (see page 1671)

1273 https://github.com/mesosphere/konvoy-image-builder/tree/main/overrides

9.9.5.3.2 Image Overrides

When using KIB to create an OS image that is compliant with DKP, the parameters for building and configuring the image are included in the file located in `images/<builder-type>/<os-version.yaml` where <builder-type> is infrastructure provider specific such as `ami` or `ova`.

Run the command for those files:

```
./konvoy-image build images/<builder-type>/{os}-{version}.yaml
```

Azure only

Azure also requires additional flags: `--client-id --tenant-id` as defined in Using KIB with Azure (see page 1642).

The YAML for each provider and OS is also located in GitHub[1274].

Although there are several parameters specified by default in the Packer templates for each provider, it is possible to **override** the default values with flags and override files.

9.9.5.3.2.1 Using Flags

To execute KIB with specific flags to override the following values:

1. the source AMI (`--source-ami`)
2. AMI region (`--ami-regions`)
3. AWS EC2 instance type (`--aws-instance-type`)

For a comprehensive list of these flags, please run:

```
./konvoy-image build --help
```

9.9.5.3.2.2 Using Override Files

Another option is by creating a file with the parameters to be overridden and specify the `--overrides` flag as shown below:

1274 https://github.com/mesosphere/konvoy-image-builder/tree/7447074a6d910e71ad2e61fc4a12820d073ae5ae/images

```
./konvoy-image build images/<builder-type>/<os>-<version>.yaml --overrides
overrides.yaml
```

While CLI flags can be used in combination with override files, CLI flags take priority over any override files.

Create a Custom Override Examples

AWS Example:

For example, when using the AWS Packer builder to override an image with another image, create an override file and set the `source_ami` under the packer key. This overrides the image search and forces the use of the specified `source_ami`.

Create your AWS override file `overrides-source-ami.yaml`:

```
---
packer:
  source_ami: "ami-0123456789"
```

After creating the override file for our `source_ami`, we can pass our override file by using the `--overrides` flag when building our image:

```
./konvoy-image build aws images/ami/centos-7.yaml --overrides override-source-
ami.yaml
```

vSphere Example:

Create your vSphere overrides file `overrides/vcenter.yaml` and fill in the relevant details for your vSphere environment:

```
---
packer:
  vcenter_server: <FQDN of your vcenter>
  vsphere_username: <username for vcenter e.g. administrator@vsphere.local>
  vsphere_password: <password for vcenter>
  ssh_username: builder
  ssh_password: <password for your VMs builder user>
  linked_clone: false
  cluster: <vsphere cluster to use>
  datacenter: <vsphere datacenter to use>
```

```
  datastore: <vsphere datastore to use for templates>
  folder: <vsphere folder to store the template in>
  insecure_connection: "true"
  network: <a DHCP-enabled network in vcenter>
  resource_pool: <vsphere resource pool to use>
```

After creating the override file for your `source_ova`, pass your override file by using the `--overrides` flag when building your image:

```
konvoy-image build images/ova/ubuntu-2004.yaml \
  --overrides overrides/kube-version.yaml \
  --overrides overrides/vcenter.yaml
```

Next Topic

Pre-Provisioned Override Files (see page 1680)

9.9.5.3.3 Pre-Provisioned Override Files

Pre-provisioned environments require certain override files to work properly. The override files can contain **HTTP Proxy** info and other factors, including whether you want the image to be **FIPS-enabled**, **NVIDIA** optimized, and so on.

Those override files must also have a **Secret** that includes all of the overrides you wish to provide in one file.

Prior to DKP 2.6, you had to specify the proxy in the KIB override setup and then again in the `dkp create cluster` command, even though in this case they would always both use the exact same proxy setting. As of DKP 2.6, an HTTP proxy gets created from the Konvoy flags for the control plane proxy and workers proxy values. The flags in the DKP command for Pre-provisioned clusters populate a Secret automatically in the bootstrap cluster. That Secret has a known name that the Pre-provisioned controller finds and applies when it runs the KIB provisioning job. More information about this flags is found on the Pre-provisioned Use HTTP Proxy (see page 1101) page.

Proxy configuration	Flag
HTTP proxy for control plane machines	`--control-plane-http-proxy string`
HTTPS proxy for control plane machines	`--control-plane-https-proxy string`
No Proxy list for control plane machines	`--control-plane-no-proxy strings`
HTTP proxy for worker machines	`--worker-http-proxy string`
HTTPS proxy for worker machines	`--worker-https-proxy string`
No Proxy list for worker machines	`--worker-no-proxy strings`

The nodes that get Kubernetes on them through CAPPP automatically get the HTTP proxy secrets that you set using the flags. You no longer have to put the proxy info in both the overrides AND the `dkp create cluster` command as an argument.

For example, if you wish to provide an override with Docker credentials and a different source for EPEL on a CentOS7 machine, you can create a file like this:

```
image_registries_with_auth:
- host: "registry-1.docker.io"
  username: "my-user"
  password: "my-password"
  auth: ""
  identityToken: ""
```

For RHEL Pre-provisioned using GPU, provide the following additional lines of information in your override file:

`rhsm_user: ""`

`rhsm_password: ""`

See Example below:

```
image_registries_with_auth:
- host: "registry-1.docker.io"
  username: "my-user"
  password: "my-password"
  auth: ""
  identityToken: ""

rhsm_user: ""
rhsm_password: ""
```

9.9.5.3.3.1 Next Topic

Using HTTP/S Proxy with KIB Images (see page 1681)

9.9.5.3.4 Using HTTP/S Proxy with KIB Images

In some networked environments, the machines used for building images can reach the Internet, but only through an HTTP/S proxy. For DKP to operate in these networks, you need a way to specify what proxies to use (see page 1053). You can use an HTTP proxy override file to specify that proxy. When KIB is trying to install a particular OS package, it uses that proxy to reach the Internet to download that package.

> ⚠ The proxy setting specified here is NOT “baked into” the image - it is only used while the image is being built. The settings are removed before the image is finalized.

While it might seem logical to include the proxy information in the image, the reality is that many companies have multiple proxies - one perhaps for each geographical region, or maybe even a proxy per Datacenter or office. All network traffic to the Internet goes through the proxy. If you were in Germany, you would probably not want to send all your traffic to a U.S.-based proxy. Doing that would slow traffic down and consume too many network resources. If you baked the proxy settings into the image, you would have to create a separate image for each specific region. It makes more sense to create an image with no proxy included, but remember that you still need a proxy to access the Internet. Thus, when creating the cluster, (and also when installing the Kommander component of DKP), you must specify the correct proxy settings for the network environment into which you are installing the cluster. You will use the same base image for *that* cluster as one installed in an environment with different proxy settings.

See: HTTP Proxy Override Files (see page 1682)

9.9.5.3.4.1 Related Topic

Pre-Provisioned Override Files (see page 1680)

9.9.5.3.4.2 Next Topic

Private Registry in Air-gapped Override (see page 1683)

9.9.5.3.4.3 Next Step

Either navigate to the main Konvoy Image Builder (see page 1626) section of the documentation or back to your installation section:

- If using the Day 1- Basic Install instructions, proceed (or return) to that section Day 1 - Basic Installs by Infrastructure (see page 146) combinations to install and setup DKP based on your infrastructure environment provider.
- If using the Custom Install instructions, proceed (or return) to that section and select the infrastructure provider you are using: Custom Installation and Additional Infrastructure Tools (see page 1050)

9.9.5.3.4.4 HTTP Proxy Override Files

You can use an HTTP proxy configuration when creating your image. The Ansible playbooks create `systemd` drop-in files for `containerd` and `kubelet` to configure the `http_proxy`, `http_proxy`, and `no_proxy` environment variables for the service from the file `/etc/konvoy_http_proxy.conf`.

To configure a proxy for use during image creation, create a new override file and specify the following:

```
# Example override-proxy.yaml
```

```
---
http_proxy: http://example.org:8080
https_proxy: http://example.org:8081
no_proxy: example.org,example.com,example.net
```

These values are only used for the image creation. After the image is created, the Ansible playbooks remove the `/etc/konvoy_http_proxy.conf` file.

You can use the `dkp` command to configure the `KubeadmConfigTemplate` object to create this file on start-up of the image with values supplied during the `dkp` invocation. This enables using different proxy settings for image creation and runtime.

Next Topic

Pre-Provisioned Override Files (see page 1680)

OR return to cluster creation in the Custom Installation and Additional Infrastructure Tools (see page 1050) under your Infrastructure Provider.

9.9.5.3.5 Private Registry in Air-gapped Override

To create an Air-gapped registry override for Docker hub, run the following command:

```
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
 endpoint = ["https://my-local-registry.local/v2/harbor-registry","https://
registry-1.docker.io"]
```

Tell the override how to create a wildcard mirror with this command:

```
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."*"]
 endpoint = ["https://my-local-registry.local/v2/harbor-registry"]
```

9.10 Konvoy Image Builder CLI

9.10.1 Other resources:

- Main Github KIB CLI[1275]
- DKP CLI (see page 1835)

9.10.2 konvoy-image build

build and provision images

9.10.2.1 Synopsis

Build and Provision images. Specifying AWS arguments is deprecated and will be removed in a future version. Use the `aws` subcommand instead.

```
konvoy-image build <image.yaml> [flags]
```

9.10.2.2 Options

```
      --ami-groups stringArray           a list of AWS groups which are allowed use
the image, using 'all' result in a public image
      --ami-regions stringArray          a list of regions to publish amis
      --ami-users stringArray            a list AWS user accounts which are allowed
use the image
      --containerd-version string        the version of containerd to install
      --dry-run                          do not create artifacts, or delete them
after creating. Recommended for tests.
      --extra-vars strings               flag passed Ansible's extra-vars
  -h, --help                             help for build
      --instance-type string             instance type used to build the AMI; the
type must be present in the region in which the AMI is built
      --kubernetes-version string        The version of kubernetes to install.
Example: 1.21.6
      --overrides strings                a comma separated list of override YAML
files
      --packer-manifest string           provide the path to a custom packer manifest
      --packer-on-error string           [advanced] set error strategy for packer.
strategies [cleanup, abort, run-cleanup-provisioner]
      --packer-path string               the location of the packer binary (default
"packer")
      --region string                    the region in which to build the AMI
```

1275 https://github.com/mesosphere/konvoy-image-builder/tree/main/docs/cli

```
      --source-ami string                the ID of the AMI to use as the source; must
be present in the region in which the AMI is built
      --source-ami-filter-name string    restricts the set of source AMIs to ones
whose Name matches filter
      --source-ami-filter-owner string   restricts the source AMI to ones with this
 owner ID
      --work-dir string                  path to custom work directory generated by
the generate command
```

9.10.2.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.2.4 SEE ALSO

- konvoy-image build aws (see page 1685) - build and provision aws images
- konvoy-image build azure (see page 1686) - build and provision azure images
- konvoy-image build gcp (see page 1688) - build and provision gcp images
- konvoy-image build vsphere (see page 1689) - build and provision vsphere images

9.10.2.5 konvoy-image build aws

build and provision aws images

```
konvoy-image build aws <image.yaml> [flags]
```

9.10.2.5.1 Examples

```
aws --region us-west-2 --source-ami=ami-12345abcdef images/ami/centos-79.yaml
```

9.10.2.5.2 Options

```
      --ami-groups stringArray           a list of AWS groups which are allowed use
the image, using 'all' result in a public image
      --ami-regions stringArray          a list of regions to publish amis
      --ami-users stringArray            a list AWS user accounts which are allowed
use the image
```

```
      --containerd-version string        the version of containerd to install
      --dry-run                          do not create artifacts, or delete them
after creating. Recommended for tests.
      --extra-vars strings               flag passed Ansible's extra-vars
  -h, --help                             help for aws
      --instance-type string             instance type used to build the AMI; the
type must be present in the region in which the AMI is built
      --kubernetes-version string        The version of kubernetes to install.
Example: 1.21.6
      --overrides strings                a comma separated list of override YAML
files
      --packer-manifest string           provide the path to a custom packer manifest
      --packer-on-error string           [advanced] set error strategy for packer.
strategies [cleanup, abort, run-cleanup-provisioner]
      --packer-path string               the location of the packer binary (default
"packer")
      --region string                    the region in which to build the AMI
      --source-ami string                the ID of the AMI to use as the source; must
be present in the region in which the AMI is built
      --source-ami-filter-name string    restricts the set of source AMIs to ones
whose Name matches filter
      --source-ami-filter-owner string   restricts the source AMI to ones with this
 owner ID
      --work-dir string                  path to custom work directory generated by
the generate command
```

9.10.2.5.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.2.5.4 SEE ALSO

- konvoy-image build (see page 1684) - build and provision images

9.10.2.6 konvoy-image build azure

build and provision azure images

```
konvoy-image build azure <image.yaml> [flags]
```

9.10.2.6.1 Examples

```
azure --location westus2 --subscription-id <sub_id> images/azure/centos-79.yaml
```

9.10.2.6.2 Options

```
      --client-id string                     the client id to use for the build
      --cloud-endpoint string                Azure cloud endpoint. Which can be one
of [Public USGovernment China] (default "Public")
      --containerd-version string            the version of containerd to install
      --dry-run                              do not create artifacts, or delete them
after creating. Recommended for tests.
      --extra-vars strings                   flag passed Ansible's extra-vars
      --gallery-image-locations stringArray  a list of locations to publish the
image (default [westus])
      --gallery-image-name string            the gallery image name to publish the
image to
      --gallery-image-offer string           the gallery image offer to set (default
 "dkp")
      --gallery-image-publisher string       the gallery image publisher to set
(default "dkp")
      --gallery-image-sku string             the gallery image sku to set
      --gallery-name string                  the gallery name to publish the image
in (default "dkp")
  -h, --help                                 help for azure
      --instance-type string                 the Instance Type to use for the build
(default "Standard_D2s_v3")
      --kubernetes-version string            The version of kubernetes to install.
Example: 1.21.6
      --location string                      the location in which to build the
image (default "westus2")
      --overrides strings                    a comma separated list of override YAML
files
      --packer-manifest string               provide the path to a custom packer
manifest
      --packer-on-error string               [advanced] set error strategy for
 packer. strategies [cleanup, abort, run-cleanup-provisioner]
      --packer-path string                   the location of the packer binary
(default "packer")
      --resource-group string                the resource group to create the image
in (default "dkp")
      --subscription-id string               the subscription id to use for the
build
      --tenant-id string                     the tenant id to use for the build
      --work-dir string                      path to custom work directory generated
by the generate command
```

9.10.2.6.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.2.6.4 SEE ALSO

- konvoy-image build (see page 1684) - build and provision images

9.10.2.7 konvoy-image build gcp

build and provision gcp images

```
konvoy-image build gcp <image.yaml> [flags]
```

9.10.2.7.1 Examples

```
gcp ... images/gcp/centos-79.yaml
```

9.10.2.7.2 Options

```
      --containerd-version string   the version of containerd to install
      --dry-run                     do not create artifacts, or delete them after
creating. Recommended for tests.
      --extra-vars strings          flag passed Ansible's extra-vars
  -h, --help                        help for gcp
      --kubernetes-version string   The version of kubernetes to install. Example:
1.21.6
      --network string              the network to use when creating an image
      --overrides strings           a comma separated list of override YAML files
      --packer-manifest string      provide the path to a custom packer manifest
      --packer-on-error string      [advanced] set error strategy for packer.
strategies [cleanup, abort, run-cleanup-provisioner]
      --packer-path string          the location of the packer binary (default
"packer")
      --project-id string           the project id to use when storing created image
      --region string               the region in which to launch the instance
(default "us-west1")
```

```
      --work-dir string             path to custom work directory generated by the
generate command
```

9.10.2.7.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.2.7.4 SEE ALSO

- konvoy-image build (see page 1684) - build and provision images

9.10.2.8 konvoy-image build vsphere

build and provision vsphere images

```
konvoy-image build vsphere <image.yaml> [flags]
```

9.10.2.8.1 Examples

```
vsphere --datacenter dc1 --cluster zone1 --datastore nfs-store1 --network public --
template=d2iq-base-templates/d2iq-base-CentOS-7.9 images/ami/centos-79.yaml
```

9.10.2.8.2 Options

```
      --cluster string              vSphere cluster to be used. Alternatively set
host. Required value: you can pass the cluster name through an override file or image
definition file.
      --containerd-version string    the version of containerd to install
      --datacenter string           The vSphere datacenter. Required value: you can
pass the datacenter name through an override file or image definition file.
      --datastore string            vSphere datastore used to build and store the
image template. Required value: you can pass the datastore name through an override
file or image definition file.
      --dry-run                     do not create artifacts, or delete them after
creating. Recommended for tests.
      --extra-vars strings          flag passed Ansible's extra-vars
      --folder string               vSphere folder to store the image template
```

```
  -h, --help                          help for vsphere
      --host string                   vSphere host to be used. Alternatively set
cluster. Required value: you can pass the host name through an override file or image
definition file.
      --kubernetes-version string     The version of kubernetes to install. Example:
1.21.6
      --network string                vSphere network used to build image template.
Ensure the host running the command has access to this network. Required value: you
can pass the network name through an override file or image definition file.
      --overrides strings             a comma separated list of override YAML files
      --packer-manifest string        provide the path to a custom packer manifest
      --packer-on-error string        [advanced] set error strategy for packer.
strategies [cleanup, abort, run-cleanup-provisioner]
      --packer-path string            the location of the packer binary (default
"packer")
      --resource-pool string          vSphere resource pool to be used to build image
template
      --ssh-privatekey-file string    Path to ssh private key which will be used to
log into the base image template
      --ssh-publickey string          Path to SSH public key which will be copied to
the image template. Ensure to set ssh-privatekey-file or load the private key into
ssh-agent
      --ssh-username string           username to be used with the vSphere image
template
      --template string               Base template to be used. Can include folder.
<templatename> or <folder>/<templatename>. Required value: you can pass the template
name through an override file or image definition file.
      --work-dir string               path to custom work directory generated by the
generate command
```

9.10.2.8.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.2.8.4 SEE ALSO

- konvoy-image build (see page 1684) - build and provision images

9.10.3 konvoy-image completion

Generate the autocompletion script for the specified shell

9.10.3.1 Synopsis

Generate the autocompletion script for konvoy-image for the specified shell. See each sub-command's help for details on how to use the generated script.

9.10.3.2 Options

```
  -h, --help   help for completion
```

9.10.3.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.3.4 SEE ALSO

- konvoy-image completion bash (see page 1691) - Generate the autocompletion script for bash
- konvoy-image completion fish (see page 1692) - Generate the autocompletion script for fish
- konvoy-image completion powershell (see page 1693) - Generate the autocompletion script for powershell
- konvoy-image completion zsh (see page 1694) - Generate the autocompletion script for zsh

9.10.3.5 konvoy-image completion bash

Generate the autocompletion script for bash

9.10.3.5.1 Synopsis

Generate the autocompletion script for the bash shell.

This script depends on the 'bash-completion' package. If it is not installed already, you can install it via your OS's package manager.

To load completions in your current shell session:

```
source <(konvoy-image completion bash)
```

To load completions for every new session, execute once:

9.10.3.5.1.1 Linux:

```
konvoy-image completion bash > /etc/bash_completion.d/konvoy-image
```

9.10.3.5.1.2 macOS:

```
konvoy-image completion bash > $(brew --prefix)/etc/bash_completion.d/konvoy-image
```

You will need to start a new shell for this setup to take effect.

```
konvoy-image completion bash
```

9.10.3.5.2 Options

```
  -h, --help              help for bash
      --no-descriptions   disable completion descriptions
```

9.10.3.5.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.3.5.4 SEE ALSO

- konvoy-image completion (see page 1690) - Generate the autocompletion script for the specified shell

9.10.3.6 konvoy-image completion fish

Generate the autocompletion script for fish

9.10.3.6.1 Synopsis

Generate the autocompletion script for the fish shell.

To load completions in your current shell session:

```
konvoy-image completion fish | source
```

To load completions for every new session, execute once:

```
konvoy-image completion fish > ~/.config/fish/completions/konvoy-image.fish
```

You will need to start a new shell for this setup to take effect.

```
konvoy-image completion fish [flags]
```

9.10.3.6.2 Options

```
  -h, --help              help for fish
      --no-descriptions   disable completion descriptions
```

9.10.3.6.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.3.6.4 SEE ALSO

- konvoy-image completion (see page 1690) - Generate the autocompletion script for the specified shell

9.10.3.7 konvoy-image completion powershell

Generate the autocompletion script for powershell

9.10.3.7.1 Synopsis

Generate the autocompletion script for powershell.

To load completions in your current shell session:

```
konvoy-image completion powershell | Out-String | Invoke-Expression
```

To load completions for every new session, add the output of the above command to your powershell profile.

```
konvoy-image completion powershell [flags]
```

9.10.3.7.2 Options

```
  -h, --help              help for powershell
      --no-descriptions   disable completion descriptions
```

9.10.3.7.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.3.7.4 SEE ALSO

- konvoy-image completion (see page 1690) - Generate the autocompletion script for the specified shell

9.10.3.8 konvoy-image completion zsh

Generate the autocompletion script for zsh

9.10.3.8.1 Synopsis

Generate the autocompletion script for the zsh shell.

If shell completion is not already enabled in your environment you will need to enable it. You can execute the following once:

```
echo "autoload -U compinit; compinit" >> ~/.zshrc
```

To load completions in your current shell session:

```
source <(konvoy-image completion zsh)
```

To load completions for every new session, execute once:

9.10.3.8.1.1 Linux:

```
konvoy-image completion zsh > "${fpath[1]}/_konvoy-image"
```

9.10.3.8.1.2 macOS:

```
konvoy-image completion zsh > $(brew --prefix)/share/zsh/site-functions/_konvoy-image
```

You will need to start a new shell for this setup to take effect.

```
konvoy-image completion zsh [flags]
```

9.10.3.8.2 Options

```
  -h, --help              help for zsh
      --no-descriptions   disable completion descriptions
```

9.10.3.8.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.3.8.4 SEE ALSO

- konvoy-image completion (see page 1690) - Generate the autocompletion script for the specified shell

9.10.4 konvoy-image generate-docs

generate docs in path

```
konvoy-image generate-docs <PATH> [flags]
```

9.10.4.1 Examples

```
generate-docs /tmp/docs
```

9.10.4.2 Options

```
  -h, --help   help for generate-docs
```

9.10.4.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.5 konvoy-image generate

generate files relating to building images

9.10.5.1 Synopsis

Generate files relating to building images. Specifying AWS arguments is deprecated and will be removed in a future version. Use the `aws` subcommand instead.

```
konvoy-image generate <image.yaml> [flags]
```

9.10.5.2 Options

```
      --ami-groups stringArray           a list of AWS groups which are allowed use
the image, using 'all' result in a public image
      --ami-regions stringArray          a list of regions to publish amis
      --ami-users stringArray            a list AWS user accounts which are allowed
use the image
      --containerd-version string        the version of containerd to install
      --extra-vars strings               flag passed Ansible's extra-vars
  -h, --help                             help for generate
```

```
      --instance-type string             instance type used to build the AMI; the
type must be present in the region in which the AMI is built
      --kubernetes-version string        The version of kubernetes to install.
Example: 1.21.6
      --overrides strings                a comma separated list of override YAML
files
      --region string                    the region in which to build the AMI
      --source-ami string                the ID of the AMI to use as the source; must
be present in the region in which the AMI is built
      --source-ami-filter-name string    restricts the set of source AMIs to ones
whose Name matches filter
      --source-ami-filter-owner string   restricts the source AMI to ones with this
 owner ID
```

9.10.5.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.5.4 SEE ALSO

- konvoy-image generate aws (see page 1697) - generate files relating to building aws images
- konvoy-image generate azure (see page 1698) - generate files relating to building azure images

9.10.5.5 konvoy-image generate aws

generate files relating to building aws images

```
konvoy-image generate aws <image.yaml> [flags]
```

9.10.5.5.1 Examples

```
aws --region us-west-2 --source-ami=ami-12345abcdef images/ami/centos-79.yaml
```

9.10.5.5.2 Options

```
      --ami-groups stringArray           a list of AWS groups which are allowed use
the image, using 'all' result in a public image
```

```
      --ami-regions stringArray          a list of regions to publish amis
      --ami-users stringArray            a list AWS user accounts which are allowed
use the image
      --containerd-version string        the version of containerd to install
      --extra-vars strings               flag passed Ansible's extra-vars
  -h, --help                             help for aws
      --instance-type string             instance type used to build the AMI; the
type must be present in the region in which the AMI is built
      --kubernetes-version string        The version of kubernetes to install.
Example: 1.21.6
      --overrides strings                a comma separated list of override YAML
files
      --region string                    the region in which to build the AMI
      --source-ami string                the ID of the AMI to use as the source; must
be present in the region in which the AMI is built
      --source-ami-filter-name string    restricts the set of source AMIs to ones
whose Name matches filter
      --source-ami-filter-owner string   restricts the source AMI to ones with this
 owner ID
```

9.10.5.5.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.5.5.4 SEE ALSO

- konvoy-image generate (see page 1696) - generate files relating to building images

9.10.5.6 konvoy-image generate azure

generate files relating to building azure images

```
konvoy-image generate azure <image.yaml> [flags]
```

9.10.5.6.1 Examples

```
azure --location westus2 --subscription-id <sub_id> images/azure/centos-79.yaml
```

9.10.5.6.2 Options

```
      --client-id string                      the client id to use for the build
      --cloud-endpoint string                 Azure cloud endpoint. Which can be one
of [Public USGovernment China] (default "Public")
      --containerd-version string             the version of containerd to install
      --extra-vars strings                    flag passed Ansible's extra-vars
      --gallery-image-locations stringArray   a list of locations to publish the
image (default [westus])
      --gallery-image-name string             the gallery image name to publish the
image to
      --gallery-image-offer string            the gallery image offer to set (default
 "dkp")
      --gallery-image-publisher string        the gallery image publisher to set
(default "dkp")
      --gallery-image-sku string              the gallery image sku to set
      --gallery-name string                   the gallery name to publish the image
in (default "dkp")
  -h, --help                                  help for azure
      --instance-type string                  the Instance Type to use for the build
(default "Standard_D2s_v3")
      --kubernetes-version string             The version of kubernetes to install.
Example: 1.21.6
      --location string                       the location in which to build the
image (default "westus2")
      --overrides strings                     a comma separated list of override YAML
files
      --resource-group string                 the resource group to create the image
in (default "dkp")
      --subscription-id string                the subscription id to use for the
build
      --tenant-id string                      the tenant id to use for the build
```

9.10.5.6.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.5.6.4 SEE ALSO

- konvoy-image generate (see page 1696) - generate files relating to building images

9.10.6 konvoy-image provision

provision to an inventory.yaml or hostname, note the comma at the end of the hostname

```
konvoy-image provision <inventory.yaml|hostname,> [flags]
```

9.10.6.1 Examples

```
provision --inventory-file inventory.yaml
```

9.10.6.2 Options

```
      --containerd-version string   the version of containerd to install
      --extra-vars strings          flag passed Ansible's extra-vars
  -h, --help                        help for provision
      --inventory-file string       an ansible inventory defining your infrastructure
      --kubernetes-version string   The version of kubernetes to install. Example:
1.21.6
      --overrides strings           a comma separated list of override YAML files
      --provider string             specify a provider if you wish to install
provider specific utilities
      --work-dir string             path to custom work directory generated by the
generate command
```

9.10.6.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.7 konvoy-image upload

Upload one of [artifacts]

9.10.7.1 Options

```
  -h, --help   help for upload
```

9.10.7.2 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.7.3 SEE ALSO

- konvoy-image upload artifacts (see page 1701) - upload offline artifacts to hosts defined in inventory-file

9.10.7.4 konvoy-image upload artifacts

upload offline artifacts to hosts defined in inventory-file

```
konvoy-image upload artifacts [flags]
```

9.10.7.4.1 Options

```
      --container-images-dir string   path to container images for install on remote
hosts.
      --containerd-bundle string      path to Containerd tar file for install on
remote hosts.
      --extra-vars strings            flag passed Ansible's extra-vars
  -h, --help                          help for artifacts
      --inventory-file string         an ansible inventory defining your
infrastructure (default "inventory.yaml")
      --nvidia-runfile string         path to nvidia runfile to place on remote
hosts.
      --os-packages-bundle string     path to os-packages tar file for install on
remote hosts.
      --overrides strings             a comma separated list of override YAML files
      --pip-packages-bundle string    path to pip-packages tar file for install on
remote hosts.
      --work-dir string               path to custom work directory generated by the
command
```

9.10.7.4.2 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.7.4.3 SEE ALSO

- konvoy-image upload (see page 1700) - Upload one of [artifacts]

9.10.8 konvoy-image validate

validate existing infrastructure

```
konvoy-image validate [flags]
```

9.10.8.1 Options

```
      --apiserver-port int      apiserver port (default 6443)
  -h, --help                    help for validate
      --inventory-file string   an ansible inventory defining your infrastructure
(default "inventory.yaml")
      --pod-subnet string       ip addresses used for the pod subnet (default
"192.168.0.0/16")
      --service-subnet string   ip addresses used for the service subnet (default
"10.96.0.0/12")
```

9.10.8.2 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.9 konvoy-image version

Version for konvoy-image

```
konvoy-image version [flags]
```

9.10.9.1 Options

```
  -h, --help   help for version
```

9.10.9.2 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.10 konvoy-image create-package-bundle

Build OS package bundles for air gapped installation.

```
konvoy-image create-package-bundle [flags]
```

9.10.10.1 Examples

```
./konvoy-image create-package-bundle --os redhat-8.4 --output-directory=artifacts
```

9.10.10.2 Options

```
      --container-image string      A container image to use for building the package
 bundles
      --fips                        If the package bundle should include fips
packages.
  -h, --help                        help for create-package-bundle
      --kubernetes-version string   The version of kubernetes to download packages
for.
      --os string                   The target OS you wish to create a package bundle
for. Must be one of [centos-7.9 redhat-7.9 redhat-8.4 redhat-8.6 redhat-8.8 rocky-9.1
 ubuntu-18.04 ubuntu-20.04]
      --output-directory string     The directory to place the bundle in. (default
"artifacts")
```

9.10.10.3 Options inherited from parent commands

```
      --color     enable color output (default true)
  -v, --v int     select verbosity level, should be between 0 and 6
      --verbose   enable debug level logging (same as --v 5)
```

9.10.10.4 SEE ALSO

- konvoy-image[1276] - Create, provision, and customize images for running Konvoy

1276 https://github.com/mesosphere/konvoy-image-builder/blob/main/docs/cli/konvoy-image.md

10 Upgrade DKP

The DKP upgrade represents an important step of your environment's lifecycle, as it ensures that you are up-to-date with the latest features and can benefit from the most recent improvements, enhanced cluster management, and better performance. This section describes how to upgrade your air-gapped and non-air-gapped environment to the latest version of DKP compatible with the latest Kubernetes version (see page 1706).

10.1 Prerequisite

Check what version of DKP you have downloaded currently using cli command dkp version (see page 1943).

```
dkp version
```

10.1.1 Supported Upgrade Paths:

Use this table to determine your correct upgrade path:

	Upgrading from Release…											
…to Release		**2.4.0**	**2.4.1**	**2.5.0**	**2.5.1**	**2.5.2**	**2.6.0**	**2.6.1**	**2.6.2**	**2.7.0**	**2.7.1**	**2.7.2**
	2.5.0	Yes	Yes	NA	NA	NA	NA	NA	NA	NA	NA	NA
	2.5.1	Yes	Yes	Yes	NA	NA	NA	NA	NA	NA	NA	NA
	2.5.2	Yes	Yes	Yes	Yes	NA	NA	NA	NA	NA	NA	NA
	2.6.0	No	No	Yes	Yes	Yes	NA	NA	NA	NA	NA	NA
	2.6.1	No	No	Yes	Yes	Yes	Yes	NA	NA	NA	NA	NA
	2.6.2	No	No	Yes	Yes	Yes	Yes	Yes	NA	NA	NA	NA
	2.7.0	No	No	No	No	No	Yes	Yes	Yes	NA	NA	NA

	2.7.1	No	No	No	No	No	Yes	Yes	Yes	Yes	NA	NA
	2.8.0	No	No	No	No	No	No	No	No	Yes	Yes	Yes
	2.8.1	No	No	No	No	No	No	No	No	Yes	Yes	Yes

10.1.2 Upgrade Order

Perform the upgrade sequentially, beginning with the Kommander component and then moving to upgrading clusters and CAPI components in the Konvoy component.

When upgrading your entire DKP product, the process is different depending on your environment.

- **Essential** (see page 1745): A stand-alone Management Cluster (see page 81)
- **Enterprise** (see page 1716): A multi-cluster environment (see page 79) that includes a combination of a Management cluster(s) and managed or attached workspace clusters.

- Before upgrading, we **strongly recommend** reading the release notes (see page 1946) and verifying your current setup against any possible breaking changes.
- Review the list of major Kubernetes changes[1277] that may affect your system.

10.1.3 Next Step:

Upgrade Compatibility Tables (see page 1706)

10.2 Upgrade Compatibility Tables

The latest DKP versions for Kubernetes, applications, and components are available on this page. Individual sections will have links to pages with further information if required.

10.2.1 Supported Operating Systems

The supported Operating System (OS) is important for upgrades. During an upgrade, you would need to create new AMI's or images with the Kubernetes version to which you want to upgrade. That requires selecting an OS and ensuring you have the right version. Check the Supported Infrastructure Operating Systems (see page 67) page if you have not already done so.

1277 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/214597661

10.2.2 Konvoy Image Builder

Refer to the main page (see page 1626)for a table of components versions and compatibility.

10.2.3 Kubernetes Version Supported

Below are the supported Kubernetes versions for both deployment and attachment:

10.2.3.1 Deploying Clusters Versions

The latest supported Kubernetes versions for DKP are listed below. In DKP 2.8.1, the Kubernetes version installed by default is 1.28.7. The newest and oldest Kubernetes versions used with DKP must be within one minor version.

- Latest supported Kubernetes version is at **1.28.7 for deploying clusters**
- Other Kubernetes versions are supported at **1.27.9 for deploying clusters in EKS**

Kubernetes 1.28.x enables you to benefit from the latest features and security fixes in upstream Kubernetes. This release comes with 60 enhancements that you can benefit from such as Node log access via Kubernetes API[1278], Seccomp profile defaulting, and much more.

To read more about major features in this release, visit this page[1279] and https://kubernetes.io/blog/2023/04/11/kubernetes-v1-27-release/ .

10.2.3.2 Attaching Clusters Versions

DKP 2.8.1 supports attaching clusters with the following Kubernetes versions:

Product	Compatible Kubernetes Versions
EKS	1.27.9
AKS	1.28.x
GKE	1.27.x

Attaching Kubernetes clusters with versions greater than n-1 is not recommended.

1278 https://github.com/kubernetes/enhancements/issues/2258
1279 https://github.com/kubernetes/sig-release/blob/master/releases/release-1.27/README.md

When attempting to attach a cluster with a lower Kubernetes version than the DKP default, you need to build and use an image with that lower version of Kubernetes. See Konvoy Image Builder (see page 1626) for documentation on building images. Failure to do this could result in an error.

10.2.3.3 Next Step:

Upgrade Prerequisites (see page 1709)

10.2.4 Upgrade Cluster Node Pools

Upgrading a node pool involves draining the existing nodes in the node pool and replacing them with new nodes. In order to ensure minimum downtime and maintain high availability of the critical application workloads during the upgrade process, we recommend deploying Pod Disruption Budget (Disruptions[1280]) for your critical applications.

The Pod Disruption Budget will prevent any impact on critical applications as a result of misconfiguration or failures during the upgrade process.

10.2.4.1 Prerequisites:

- Deploy Pod Disruption Budget[1281] (PDB)
- Konvoy Image Builder (see page 1626) (KIB)

10.2.4.2 Steps:

1. Deploy Pod Disruption Budget for your critical applications. If your application can tolerate only one replica to be unavailable at a time, then you can set Pod disruption budget as shown in the following example. The example below is for NVIDIA GPU node pools, but the process is the same for all node pools.

Repeat this step for each additional node pool.

Create the file: `pod-disruption-budget-nvidia.yaml`

1280 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/
1281 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: nvidia-critical-app
spec:
  maxUnavailable: 1
  selector:
    matchLabels:
      app: nvidia-critical-app
```

Apply the YAML file above using the following command:

```
kubectl create -f pod-disruption-budget-nvidia.yaml
```

2. Prepare OS image for your node pool using Konvoy Image Builder (see page 1626).

10.2.4.3 Related Topics:

Enterprise: Upgrade the Management Cluster Kubernetes Version (see page 1735)

10.3 Upgrade Prerequisites

The prerequisites vary depending on your environment. Ensure you follow the correct lists for your specific configurations:

10.3.1 Prerequisites for the Kommander Component

The prerequisites for Kommander consists of two parts, the first section for all environments, and the subsections depend on air-gapped or non-air-gapped, and optional applications. Ensure you follow the correct section for your specific environment type.

10.3.1.1 All Kommander Environments

For all Kommander environments, regardless of network type and applications, complete these prerequisites:

- Before upgrading, we **strongly recommend** reading the release notes (see page 1946) and verifying your current setup against any possible breaking changes.
- Review the Components and Applications (see page 1950) versions that are part of this upgrade.
- Ensure your applications you are running will be compatible with Kubernetes v1.27.x.
- If you have attached clusters, ensure your applications you are running will be compatible with Kubernetes v1.27.x.
- Review the list of major Kubernetes changes (see page 1949) that may affect your system.
- Ensure you are attempting to run a supported DKP upgrade (see page 0).
- **REQUIRED** Before upgrading, create an on-demand backup (see page 904) of your current configuration with Velero.
- Download (see page 76) and install the supported DKP CLI binary of this release[1282] on your computer.

The **remaining prerequisites** are necessary if you have one of the following environments or additions to basic Kommander. Otherwise, proceed to the prerequisites for the Konvoy component.

10.3.1.1.1 Air-gapped Kommander Environments Only

If you operate Kommander in an air-gapped environment, complete these prerequisites:

- There are several sets of images (see page 1712) you will need to push to your local registry:
  - The Kommander component - kommander-image-bundle-v2.7.0.tar
  - The Konvoy component for Kubernetes cluster - konvoy-image-bundle-v2.7.0.tar
  - **Optional:** Catalog Applications (Enterprise only) - dkp-catalog-applications-image-bundle-v2.7.0.tar
- **For air-gapped environments with catalog applications:** Ensure you have updated your catalog repository before upgrading. The catalog repository should contain the Docker registry with the DKP catalog application (see page 700) images and a charts bundle file containing DKP catalog applications (see page 0) charts.

10.3.1.1.2 DKP Workspace Catalog Applications Only

If you use any of the DKP Workspace Catalog Applications, complete these prerequisites:

- Ensure your clusters run on a supported Kubernetes version for this release of DKP (see page 1948), and that said Kubernetes version is also compatible with your catalog application version (see page 0).
- For customers with an **Enterprise license** (see page 89) and a multi-cluster environment, we recommend keeping all clusters on the same Kubernetes version. This ensures your DKP catalog application can run on all clusters in a given workspace.
- Konvoy installs a new Kubernetes version with each upgrade, therefore it is important that your Catalog application (see page 692) is compatible with the Kubernetes version that comes with this release of DKP.

1282 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/214564869

If your current Catalog application version is not compatible with this release's Kubernetes version, upgrade the application to a compatible version BEFORE upgrading the Konvoy component on *Managed clusters* (see page 80) or BEFORE upgrading the Kubernetes version on *Attached clusters* (see page 80).

- **For air-gapped environments with catalog applications:** Ensure you have updated your catalog repository before upgrading. The catalog repository should contain the Docker registry with the DKP catalog application (see page 700) images and a charts bundle file containing DKP catalog applications (see page 0) charts.

10.3.2 Prerequisites for the Konvoy Component

The prerequisites for Konvoy also consists of two parts, the first section for all environments, and the subsections depend on your network type, air-gapped or non-air-gapped, and optional applications. Ensure you follow the correct section for your specific environment type.

10.3.2.1 All Konvoy Environments

For all Konvoy environments, regardless of network type and applications, complete these prerequisites:

- Check what version of DKP you have downloaded using the CLI command dkp version (see page 1943).
- Set the appropriate environment variables:
  - For Air-gapped, download the required bundles at our support site[1283].
  - For Azure, set the required environment variables (see page 1296).
  - For AWS and EKS (see page 1156), set the required environment variables (see page 295).
  - For vSphere, set the required environment variables (see page 1356).
  - For GCP, set the required environment variables (see page 1487).
- **vSphere only:** If you want to resize your disk, ensure you have reviewed Create a vSphere Base OS Image (see page 1652).

10.3.2.1.1 Air-gapped Konvoy Environments Only

If you operate Konvoy in an air-gapped environment, complete these prerequisites:

- Check what version of DKP you have downloaded currently using cli command dkp version[1284].
- Ensure that all platform applications in the management cluster have been upgraded to avoid compatibility issues with the Kubernetes version[1285] included in this release. This is done

1283 https://support.d2iq.com/hc/en-us
1284 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/213157688
1285 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/214630401

automatically when upgrading Kommander (see page 1709), so ensure that you upgrade Kommander prior to upgrading Konvoy.
- For air-gapped environments, seed the registry as explained here: Upgrade: For Air-gapped Environments Only (see page 1712)

10.3.3 Third-Party and Open Source Attributions

See https://d2iq.com/legal/3rd for a list of third-party and open source attributions.

10.3.3.1 Next Steps:

Depending on your license type, you will follow the relevant link.

- Upgrade DKP Enterprise (see page 1716)
- Upgrade DKP Essential (see page 1745)

10.4 Upgrade: For Air-gapped Environments Only

Because air-gapped environments do not have direct access to the Internet, you must download, extract and load several required images to your local container registry (see page 1606), before installing or upgrading DKP. The information below will be covered as a step during either **Enterprise** or **Essential** Upgrade steps, but feel free to familiarize yourself with the concept below if desired. Otherwise, depending on your license type, follow the relevant link to begin upgrading:

- Upgrade DKP Enterprise (see page 1716)
- Upgrade DKP Essential (see page 1745)

10.4.1 Overview of Seeding the Registry for Air-gapped Environment

10.4.2 Download all Images for Air-gapped Deployments

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for prerequisites to download and then how to push the necessary images to this registry.

1. Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`) to load registry images as explained below.
2. Connectivity with clusters attaching to the management cluster is required:
    - Both management and attached clusters must be able to connect to the local registry.
    - The management cluster must be able to connect to all attached cluster's API servers.
    - The management cluster must be able to connect to any load balancers created for platform services on the management cluster.

10.4.3 Extract Air-gapped Images and Set Variables

Follow these steps to **extract** the air-gapped image bundles into your private registry:

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

   ```
   cd dkp-v2.8.1
   ```

3. Set an environment variable with your registry address:

   ```
   export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   ```

10.4.3.1 Only Pre-provisioned: Load Images for Deployments - Konvoy

For **Pre-provisioned air-gapped environments only**, you must run `konvoy-image upload artifacts` to copy the artifacts (see page 1120) onto the cluster hosts **before** you begin the Upgrade the CAPI Components (see page 1731) process later in the upgrade steps.

1. The Kubernetes image bundle will be located in `kib/artifacts/images` and you will want to verify image and artifacts.
   a. Verify the image bundles exist in `kib/artifacts/images`:

      ```
      $ ls kib/artifacts/images/
      kubernetes-images-1.28.7-d2iq.1.tar kubernetes-images-1.28.7-d2iq.1-
      fips.tar
      ```

   b. Verify the artifacts for your OS exist in the `kib/artifacts/` directory and export the appropriate variables:

      ```
      $ ls kib/artifacts/
      ```

```
1.28.7_centos_7_x86_64.tar.gz       1.28.7_redhat_8_x86_64_fips.tar.gz
containerd-1.6.28-d2iq.1-rhel-7.9-x86_64.tar.gz        containerd-1.6.28-
d2iq.1-rhel-8.6-x86_64_fips.tar.gz pip-packages.tar.gz
1.28.7_centos_7_x86_64_fips.tar.gz  1.28.7_rocky_9_x86_64.tar.gz
containerd-1.6.28-d2iq.1-rhel-7.9-x86_64_fips.tar.gz   containerd-1.6.28-
d2iq.1-rocky-9.0-x86_64.tar.gz
1.28.7_redhat_7_x86_64.tar.gz       1.28.7_ubuntu_20_x86_64.tar.gz
containerd-1.6.28-d2iq.1-rhel-8.4-x86_64.tar.gz        containerd-1.6.28-
d2iq.1-rocky-9.1-x86_64.tar.gz
1.28.7_redhat_7_x86_64_fips.tar.gz  containerd-1.6.28-d2iq.1-centos-7.9-
x86_64.tar.gz       containerd-1.6.28-d2iq.1-rhel-8.4-x86_64_fips.tar.gz
containerd-1.6.28-d2iq.1-ubuntu-20.04-x86_64.tar.gz
1.28.7_redhat_8_x86_64.tar.gz       containerd-1.6.28-d2iq.1-centos-7.9-
x86_64_fips.tar.gz  containerd-1.6.28-d2iq.1-rhel-8.6-x86_64.tar.gz
images
```

c. Set the bundle values with the name from the private registry location:

```
export OS_PACKAGES_BUNDLE=<name_of_the_OS_package>
export CONTAINERD_BUNDLE=<name_of_the_containerd_bundle>
```

For example, for RHEL 8.4 you would set:

```
export OS_PACKAGES_BUNDLE=1.28.7_redhat_8_x86_64.tar.gz
export CONTAINERD_BUNDLE=containerd-1.6.28-d2iq.1-rhel-8.4-x86_64.tar.gz
```

2. Upload the artifacts onto cluster hosts:

```
konvoy-image upload artifacts \
    --container-images-dir=./kib/artifacts/images/ \
    --os-packages-bundle=./kib/artifacts/${OS_PACKAGES_BUNDLE} \
    --containerd-bundle=./kib/artifacts/${CONTAINERD_BUNDLE} \
    --pip-packages-bundle=./kib/artifacts/pip-packages.tar.gz
```

10.4.3.2 Load Images to your Private Registry - Konvoy

Before creating or upgrading a Kubernetes cluster, you need to load the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, refer to Local Registry Tools (see page 1606) page for more information.

Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

10.4.3.3 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

10.4.3.4 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME}
--to-registry-password=${REGISTRY_PASSWORD}
```

10.4.3.5 Next Step:

Depending on your license type, you will follow the relevant link:

- Upgrade DKP Enterprise (see page 1716)

- Upgrade DKP Essential (see page 1745)

10.5 Upgrade DKP Enterprise

Enterprise Gov Advanced

10.5.1 Steps to upgrade DKP via CLI

Before you begin the upgrade, ensure you review the Upgrade Prerequisites (see page 1709).

This section describes how to upgrade your DKP clusters to the latest DKP version in an Enterprise, multi-cluster, environment. The DKP upgrade represents an important step of your environment's lifecycle, as it ensures that you are up-to-date with the latest features and can benefit from the most recent improvements, enhanced cluster management, and better performance. If you are on this scenario of the upgrade section of documentation, you have an Enterprise License (see page 85). For an Essential (see page 85)license, refer to the DKP Upgrade Essential (see page 1745) section instead.

Ensure that all platform applications in the management cluster have been upgraded to avoid compatibility issues with the Kubernetes version (see page 1706) included in this release. This is done automatically when upgrading Kommander, so ensure that you follow these sections in order to upgrade the Kommander component prior to upgrading the Konvoy component.

10.5.2 Overview of Steps in Order

- Enterprise: For Air-gapped Environments Only (see page 1717)
- Enterprise: Upgrade the Management Cluster and Platform Applications (see page 1719)
- Enterprise: Upgrade Platform Applications on Managed and Attached Clusters (see page 1722)
- Enterprise: Upgrade Workspace DKP Catalog Applications (see page 1724)
- Enterprise: Upgrade Project Catalog Applications (see page 1729)
- Enterprise: Upgrade Custom Applications (see page 1731)
- Enterprise: Upgrade the Management Cluster CAPI Components (see page 1731)
- Enterprise: Upgrade the Management Cluster Core Addons (see page 1732)
- Enterprise: Upgrade the Management Cluster Kubernetes Version (see page 1735)
- Enterprise: Upgrade Managed Clusters (see page 1739)
- Enterprise: Upgrade images used by Catalog Applications (see page 1742)

10.5.3 Next Step:

If you are in an air-gapped environment, be sure to select that choice to find out how to load your local registry first. Otherwise, your next step will be Upgrade the Management Cluster and Platform Applications (see page 1719).

- Enterprise: For Air-gapped Environments Only (see page 1717)

10.5.4 Enterprise: For Air-gapped Environments Only

Enterprise Gov Advanced

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

- Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.7.0_linux_amd64.tar.gz`)
- Connectivity with clusters attaching to the management cluster:
  - Both management and attached clusters must be able to connect to the local registry.
  - The management cluster must be able to connect to all attached cluster's API servers.
  - The management cluster must be able to connect to any load balancers created for platform services on the attached cluster.

Note: You can also attach clusters when there are networking restrictions between the management cluster and attached cluster.
See Attach a Cluster with Networking Restrictions (see page 804) for more information.

10.5.4.1 Extract Air-gapped Images and Set Variables

Follow these steps to **extract** the air-gapped image bundles into your private registry:

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

```
cd dkp-v2.8.1
```

3. Set an environment variable with your registry address:

```
export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
export REGISTRY_USERNAME=<username>
export REGISTRY_PASSWORD=<password>
```

10.5.4.1.1 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

10.5.4.1.2 Load Images to your Private Registry - DKP Catalog Applications

Optional: This step is required only if you have an Enterprise license.

For **DKP Catalog Applications**, also perform this image load:

Run the following command to load the `dkp-catalog-applications` image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/dkp-catalog-applications-image-bundle-
v2.8.1.tar --to-registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME}
--to-registry-password=${REGISTRY_PASSWORD}
```

10.5.4.1.3 Load Images to your Private Registry - Konvoy

Before creating or upgrading a Kubernetes cluster, you need to load the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, refer to Local Registry Tools (see page 1606) page for more information.

Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

10.5.4.1.4 Next Step:

Enterprise: Upgrade the Management Cluster and Platform Applications (see page 1719)

10.5.5 Enterprise: Upgrade the Management Cluster and Platform Applications

Enterprise Gov Advanced

This section describes how to upgrade your Kommander Management cluster and all Platform Applications to their supported versions in air-gapped, non-air-gapped and on-prem environments. To prevent compatibility issues, you must first upgrade Kommander on your Management Cluster before upgrading to DKP.

It is important you upgrade Kommander BEFORE upgrading the Kubernetes version (or Konvoy version for Managed Konvoy clusters) in attached clusters. This ensures that any changes required for new or changed Kubernetes API's are already present.

10.5.5.1 Upgrade Kommander

10.5.5.1.1 Prerequisites

- Use the `--kubeconfig=${CLUSTER_NAME}.conf` flag or set the `KUBECONFIG` environment variable to ensure that you **upgrade Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 106).

- If you have DKP Insights installed, ensure you **uninstall** it before upgrading DKP. See DKP Insights Upgrade to 1.0.0 (DKP 2.7.0)[1286] for more information.

Limited availability of DKP resources

- If you have configured a **custom domain**, running the `upgrade` command could result in an inaccessibility of your services via your custom domain for a few minutes.
- The **DKP UI** and other APIs may be inconsistent or unavailable until the upgrade is complete.

10.5.5.1.2 Upgrade

Use the DKP CLI to upgrade Kommander and all the Platform Applications in the Management Cluster:

For non-air-gapped environments

```
dkp upgrade kommander
```

If you want to disable the AI Navigator, add the flag `--disable-appdeployments ai-navigator-app` to this command.

The output looks similar to this:

```
✓ Ensuring upgrading conditions are met
✓ Ensuring application definitions are updated
✓ Ensuring helm-mirror implementation is migrated to ChartMuseum
...
```

After the upgrade, if you have DKP Catalog Applications deployed, proceed to https://d2iq.atlassian.net/wiki/spaces/DENT/pages/221577412/Enterprise+Upgrade+Workspace+DKP+Catalog+Applications#Update-the-GitRepository-(for-non-air-gapped-environments) (see page 0).

For air-gapped environments

```
dkp upgrade kommander \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.0.tar.gz \
--kommander-applications-repository ./application-repositories/kommander-applications-v2.8.0.tar.gz
```

1286 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/247169025

The output looks similar to this:

```
✓ Ensuring upgrading conditions are met
✓ Ensuring application definitions are updated
✓ Ensuring helm-mirror implementation is migrated to ChartMuseum
...
```

For air-gapped environments with DKP Catalog Applications

```
dkp upgrade kommander \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.8.0.tar.gz \
--charts-bundle ./application-charts/dkp-catalog-applications-charts-bundle-v2.8.0.ta
r.gz \
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.8.0.tar.gz
```

The output looks similar to this:

```
✓ Ensuring upgrading conditions are met
✓ Ensuring application definitions are updated
✓ Ensuring helm-mirror implementation is migrated to ChartMuseum
...
```

After the upgrade, if you have DKP Catalog Applications deployed, proceed to https://d2iq.atlassian.net/wiki/spaces/DENT/pages/221577412/Enterprise+Upgrade+Workspace+DKP+Catalog+Applications#Update-the-GitRepository-(for-air-gapped-environments) (see page 0).

10.5.5.1.2.1 Troubleshooting

1. If the upgrade fails, run the following command to get more information on the upgrade process:

   ```
   dkp upgrade kommander -v 6
   ```

2. If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

   ```
   kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"
   op": "replace", "path": "/spec/suspend", "value": true}]'
   kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"
   op": "replace", "path": "/spec/suspend", "value": false}]'
   ```

10.5.5.1.3 Next Step:

https://d2iq.atlassian.net/wiki/spaces/DENT/pages/221610095/Enterprise+Upgrade+Additional+Workspaces (see page 1722)

10.5.6 Enterprise: Upgrade Platform Applications on Managed and Attached Clusters

Enterprise Gov Advanced

For the Management Cluster (see page 80) (and the `kommander` workspace namespace), `dkp upgrade` handles all Platform applications; no other steps are necessary. However, for managed (see page 80) or attached clusters (see page 80), you MUST manually upgrade the Platform applications by upgrading the workspace.

> ⚠ If you are upgrading your Platform applications as part of the DKP upgrade (see page 1705), upgrade your Platform applications on any additional Workspaces before proceeding with the Konvoy upgrade. Some applications in the previous release are not compatible with the Kubernetes version[1287] of this release, and upgrading Kubernetes is part of the DKP Konvoy upgrade process.

10.5.6.1 Prerequisites

Before you begin, you must:

- Set the `WORKSPACE_NAMESPACE` environment variable to the name of the workspace's namespace where the cluster is attached:

  ```
  export WORKSPACE_NAMESPACE=<workspace_namespace>
  ```

- Set the `WORKSPACE_NAME` environment variable to the name of the workspace where the cluster is attached:

  ```
  export WORKSPACE_NAME=<workspace_name>
  ```

1287 https://d2iq.atlassian.net/wiki/pages/createpage.action?fromPageId=221610095&linkCreation=true&spaceKey=DENT&title=DKP+2.6.0+Supported+Kubernetes+Versions

10.5.6.2 Upgrade Platform Applications from the CLI

10.5.6.2.1 Execute the DKP Upgrade Command

Upgrade all Platform applications in a workspace and its projects to the same version as the platform applications running on the management cluster:

```
dkp upgrade workspace ${WORKSPACE_NAME}
```

An output similar to this appears:

```
✓ Ensuring HelmReleases are upgraded on clusters in namespace "${WORKSPACE_NAME}"
...
```

10.5.6.2.2 Troubleshooting

If the upgrade fails or times out, retry the command with higher verbosity to get more information on the upgrade process:

```
dkp upgrade workspace ${WORKSPACE_NAME} -v 4
```

If you find any `HelmReleases` in a “broken” release state, such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

```
kubectl -n ${WORKSPACE_NAMESPACE} patch helmrelease <HELMRELEASE_NAME> --type='json'
 -p='[{"op": "replace", "path": "/spec/suspend", "value": true}]'
kubectl -n 4{WORKSPACE_NAMESPACE} patch helmrelease <HELMRELEASE_NAME> --type='json'
 -p='[{"op": "replace", "path": "/spec/suspend", "value": false}]'
```

10.5.6.3 Next Step:

Enterprise: Upgrade Workspace Catalog Applications (see page 1724)

10.5.7 Enterprise: Upgrade Workspace DKP Catalog Applications

Upgrade catalog applications using the CLI and UI

Enterprise Gov Advanced

10.5.7.1 Update the DKP Catalog Applications GitRepository

Follow the instructions in this section to update the GitRepository `dkp-catalog-applications`.

For air-gapped environments

10.5.7.2 Update the GitRepository (for air-gapped environments)

Update the GitRepository for `dkp-catalog-applications`.

> ⚠ For the following section, ensure you modify the **most recent** `kommander.yaml` configuration file. It must be the file that reflects the current state of your environment. Reinstalling Kommander with an outdated `kommander.yaml` overwrites the list of platform applications that are currently running in your cluster.

In the `kommander.yaml` you are currently using for your environment, update the DKP Catalog Applications by setting the correct DKP version:

```
...
# The list of enabled/disabled apps here should reflect the current state of the
environment, including configuration overrides!
...
catalog:
  repositories:
    - name: dkp-catalog-applications
      labels:
        kommander.d2iq.io/project-default-catalog-repository: "true"
        kommander.d2iq.io/workspace-default-catalog-repository: "true"
        kommander.d2iq.io/gitapps-gitrepository-type: "dkp"
      path: ./dkp-catalog-applications-v2.8.0.tar.gz # modify this version to match
the DKP upgrade version
...
```

Refresh the `kommander.yaml` to apply the updated tarball:

Ensure the `kommander.yaml` is the Kommander Installer Configuration file you are currently using for your environment. Otherwise, your configuration will be overwritten and previous configuration lost.

```
dkp install kommander --installer-config kommander.yaml
```

For a final cleanup of Spark, delete its AppDeployment from all workspaces where it was deployed:

1. Execute the following command to get the `WORKSPACE_NAMESPACE` of your workspace:

   ```
   dkp get workspaces
   ```

   And copy the values under the `NAMESPACE` column for your workspace.

2. Export the `WORKSPACE_NAMESPACE` variable:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

3. Run the following command to delete the `spark-operator` AppDeployment:

   ```
   kubectl delete AppDeployment <spark operator appdeployment name> -n ${WORKSPACE_NAMESPACE}
   ```

For non-air-gapped environments

10.5.7.2.1 Update the GitRepository (for non-air-gapped environments)

Upgrade the workspace Catalog Applications in `kommander` on each additional workspace on which catalog apps are deployed. In air-gapped scenarios, the images are included in the air-gapped bundle. In non-air-gapped scenarios, refresh the Git Repository as shown below.

Update the GitRepository with the tag of your updated DKP version on the `kommander` workspace:

```
kubectl patch gitrepository -n kommander dkp-catalog-applications --type merge --patch '{"spec":{"ref":{"tag":"v2.8.0"}}}'
```

This command updates the catalog application repositories for all workspaces.

For any additional Catalog `GitRepository` resources created outside the `kommander.yaml` configuration (e.g. such as in a project or workspace namespace), follow these steps:
Set the `WORKSPACE_NAMESPACE` environment variable to the namespace of the workspace:

```
export WORKSPACE_NAMESPACE=<workspace namespace>
```

Update the GitRepository for the additional workspace:

```
kubectl patch gitrepository -n ${WORKSPACE_NAMESPACE} dkp-catalog-applications --type
merge --patch '{"spec":{"ref":{"tag":"v2.8.0"}}}'
```

For a final cleanup of Spark, delete its AppDeployment from all workspaces where it was deployed:

1. Execute the following command to get the `WORKSPACE_NAMESPACE` of your workspace:

   ```
   dkp get workspaces
   ```

   And copy the values under the `NAMESPACE` column for your workspace.

2. Export the `WORKSPACE_NAMESPACE` variable:

   ```
   export WORKSPACE_NAMESPACE=<WORKSPACE_NAMESPACE>
   ```

3. Run the following command to delete the `spark-operator` AppDeployment:

   ```
   kubectl delete AppDeployment <spark operator appdeployment name> -n $
   {WORKSPACE_NAMESPACE}
   ```

10.5.7.3 Upgrade Catalog Applications

Before upgrading also, keep in mind the distinction between Platform applications and Catalog applications. Platform applications are deployed and upgraded as a set for each cluster or workspace. Catalog applications are deployed separately, so that you can deploy and upgrade them individually for each project.

10.5.7.3.1 Upgrade with UI

Follow these steps to upgrade an application from the DKP UI:

1. From the top menu bar, select your target workspace.
2. Select **Applications** from the sidebar menu.
3. Select the three dot button from the bottom-right corner of the desired application tile, and then select **Edit**.
4. Select the **Version** drop-down, and select a new version. This drop-down will only be available if there is a newer version to upgrade to.
5. Select **Save**.
6. Repeat this process for on each workspace where you have deployed the application.

10.5.7.3.2 Upgrade with CLI

Please note that the below commands are using the workspace **name** and not namespace. You can retrieve the workspace name by running the command:
`dkp get workspaces`
To view a list of the deployed `apps` to your workspace, you can run the command:
`dkp get appdeployments --workspace=<workspace-name>`

1. To see what app(s) and app versions are available to upgrade, run the following command:
   NOTE: You can reference the app version by going into the app name (e.g. `<APP ID>-<APP VERSION>`)

   ```
   kubectl get apps -n ${WORKSPACE_NAMESPACE}
   ```

   You can also use this command to display the apps and app versions, for example:

   ```
   kubectl get apps -n ${WORKSPACE_NAMESPACE} -o jsonpath='{range .items[*]}
   {@.spec.appId}{"----"}{@.spec.version}{"\n"}{end}'
   ```

   Below is an example of an output that shows the different apps and apps versions.

   ```
   kafka-operator----0.20.0
   kafka-operator----0.20.2
   ```

```
kafka-operator----0.23.0-dev.0
zookeeper-operator----0.2.13
zookeeper-operator----0.2.14
```

2. Run the following command to upgrade an application from the DKP CLI:

   ```
   dkp upgrade catalogapp <appdeployment-name> --workspace=${WORKSPACE_NAME} --to-version=<version.number>
   ```

   As an example, the following command upgrades the Kafka Operator application, named `kafka-operator-abc`, in a workspace to version `0.25.1`:

   ```
   dkp upgrade catalogapp kafka-operator-abc --workspace=${WORKSPACE_NAME} --to-version=0.25.1
   ```

3. Repeat this process for on each workspace where you have deployed the application.

Platform applications cannot be upgraded on a one-off basis, and must be upgraded in a single process for each workspace. If you attempt to upgrade a platform application with these commands, you receive an error and the application is not upgraded.

As of DKP 2.8, the versions of Catalog applications other than these listed below are deprecated:

- `kafka-operator-0.25.1`
- `zookeeper-operator-0.2.16-dkp.1`

If you plan on upgrading to DKP 2.8 or above, ensure that you upgrade these applications to the latest compatible version.

- To find what versions of applications are available for upgrade, use `kubectl`:

  ```
  kubectl get apps -n ${WORKSPACE_NAMESPACE}
  ```

For more information, see Workspace DKP Catalog Applications | Workspace DKP Catalog Applications (see page 692).

To ensure you do not install images with known CVE's, you should specify a custom image for `kafka` and `zookeeper` by following these instructions:

- For `kafka` :https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919406/Kafka+in+a+Workspace#Using-a-custom-image-for-a-zookeeper-cluster (see page 0)
- For `zookeeper` :https://d2iq.atlassian.net/wiki/spaces/DENT/pages/29919470/Zookeeper+Operator+in+a+Workspace#Using-a-custom-image-for-a-zookeeper-cluster (see page 0)

10.5.7.4 Next Step:

Enterprise: Upgrade Project Catalog Applications (see page 1729)

10.5.8 Enterprise: Upgrade Project Catalog Applications

Enterprise Gov Advanced

Before upgrading your catalog applications, verify the current and supported versions of the application. Also, keep in mind the distinction between Platform applications and Catalog applications. Platform applications are deployed and upgraded as a set for each cluster or workspace. Catalog applications are deployed separately, so that you can deploy and upgrade them individually for each project.

Catalog applications must be upgraded to the latest version BEFORE upgrading the Konvoy components for Managed clusters or Kubernetes version for attached clusters.

10.5.8.1 Upgrade with the UI

Follow these steps to upgrade an application from the DKP UI:

1. From the top menu bar, select your target workspace.
2. From the side menu bar, select **Projects**.
3. Select your target project.
4. Select **Applications** from the project menu bar.
5. Select the three dot button from the bottom-right corner of the desired application tile, and then select **Edit**.
6. Select the **Version** drop-down, and select a new version. This drop-down will only be available if there is a newer version to upgrade to.
7. Select **Save**.

10.5.8.2 Upgrade with the CLI

1. To see what app(s) and app versions are available to upgrade, run the following command:
   NOTE: You can reference the app version by going into the app name (e.g. `<APP ID>-<APP VERSION>`)

   ```
   kubectl get apps -n ${PROJECT_NAMESPACE}
   ```

   You can also use this command to display the apps and app versions, for example:

   ```
   kubectl get apps -n ${PROJECT_NAMESPACE} -o jsonpath='{range .items[*]}
   {@.spec.appId}{"----"}{@.spec.version}{"\n"}{end}'
   ```

   Below is an example of an output that shows the different apps and apps versions.

   ```
   zookeeper-operator----0.2.13
   zookeeper-operator----0.2.14
   zookeeper-operator----0.2.15
   ```

2. Run the following command to upgrade an application from the DKP CLI:

   ```
   dkp upgrade catalogapp <appdeployment-name> --workspace=my-workspace --
   project=my-project --to-version=<version.number>
   ```

As an example, the following command upgrades the Zookeeper Operator application, named `zookeeper-operator-abc` , in a workspace to version `0.2.15` :

```
dkp upgrade catalogapp zookeeper-operator-abc --workspace=my-workspace --to-
version=0.2.15
```

> Platform applications cannot be upgraded on a one-off basis, and must be upgraded in a single process for each workspace. If you attempt to upgrade a platform application with these commands, you receive an error and the application is not upgraded.

10.5.8.3 Next Step:

Enterprise: Upgrade Custom Applications (see page 1731)

10.5.9 Enterprise: Upgrade Custom Applications

Enterprise Gov Advanced

10.5.9.1 Verify the compatibility of Custom Applications with the current Kubernetes version

We recommend upgrading your Custom Applications to the latest compatible version as soon as possible. Since Custom Applications are not created, maintained or supported by D2iQ, you must upgrade them manually.

Ensure you validate any Custom Applications you run for compatibility issues against the Kubernetes version (see page 1946) in the new release. If the Custom Application's version is not compatible with the Kubernetes version, do not continue with the Konvoy upgrade. Otherwise, your custom Applications may stop running.

After you have ensured your Custom Applications are compatible with the current Kubernetes version, return to the DKP Upgrade overview (see page 0) documentation, to review the next steps depending on your environment and license type.

10.5.9.2 Next Step:

Enterprise: Upgrade the Management Cluster CAPI Components (see page 1731)

10.5.10 Enterprise: Upgrade the Management Cluster CAPI Components

Upgrade the CAPI Components on the Management cluster.

Enterprise Gov Advanced

For a Pre-provisioned air-gapped environment only, ensure you have uploaded the artifacts (see page 1712).

10.5.10.1 Upgrade the CAPI Components

New versions of DKP come pre-bundled with newer versions of CAPI, newer versions of infrastructure providers, or new infrastructure providers. When using a new version of the DKP CLI, upgrade all of these components first.

If you are running on more than one management cluster, you must upgrade the CAPI components on each of these clusters.

Ensure your `dkp` configuration references the management cluster where you want to run the upgrade by setting the `KUBECONFIG` environment variable, or using the `--kubeconfig` flag, in accordance with Kubernetes conventions[1288].

Execute the upgrade command for the CAPI components.

If you created CAPI components using flags to specify values, use those same flags during **Upgrade** to preserve existing values while setting additional values.

- Refer to dkp create cluster aws (see page 1856) for flag descriptions for `--with-aws-bootstrap-credentials` and `--aws-service-endpoints`
- Refer to the HTTP section for details: Create CAPI Components with HTTP/HTTPS Proxy Settings (see page 1055)

```
dkp upgrade capi-components
```

The output resembles the following:

```
✓ Upgrading CAPI components
✓ Waiting for CAPI components to be upgraded
✓ Initializing new CAPI components
```

If the upgrade fails, review the prerequisites section and ensure that you've followed the steps in the DKP Upgrade (see page 1705) overview. Furthermore, ensure you have adhered to the Prerequisites at the top of this page.

10.5.10.2 Next Step:

Enterprise: Upgrade the Management Cluster Core Addons (see page 1732)

10.5.11 Enterprise: Upgrade the Management Cluster Core Addons

Upgrade the Core Addons on the **Management Cluster**.

Enterprise Gov Advanced

1288 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

For Pre-provisioned air-gapped environments only, you must run `konvoy-image upload artifacts` to copy the artifacts (see page 1120) onto the cluster hosts **before** you begin the Upgrade the CAPI Components section below.

```
konvoy-image upload artifacts \
    --container-images-dir=./artifacts/images/ \
    --os-packages-bundle=./artifacts/$OS_PACKAGES_BUNDLE \
    --containerd-bundle=artifacts/$CONTAINERD_BUNDLE \
    --pip-packages-bundle=./artifacts/pip-packages.tar.gz
```

To install the core addons, DKP relies on the `ClusterResourceSet` Cluster API feature[1289]. In the CAPI component upgrade, we deleted the previous set of outdated global `ClusterResourceSets` because in past releases, some addons were installed using a global configuration. In order to support individual cluster upgrades, DKP now installs all addons with a unique set of `ClusterResourceSets` and corresponding referenced resources, all named using the cluster's name as a suffix. For example: `calico-cni-installation-my-aws-cluster`.

If you have modified any of the `ClusterResourceSet` definitions, these changes will **not** be preserved when running the command `dkp upgrade addons <provider>`. You must define the cloud provider before you use the `--dry-run -o yaml` options to save the new configuration to a file and remake the same changes upon each upgrade.

Your cluster comes preconfigured with a few different core addons that provide functionality to your cluster upon creation. These include: CSI, CNI, Cluster Autoscaler, and Node Feature Discovery. New versions of DKP may come pre-bundled with newer versions of these addons.

Perform the following steps to update these addons:

1. If you have any additional managed clusters, you will need to upgrade the core addons and Kubernetes version for each one.
2. Ensure your `dkp` configuration references the management cluster where you want to run the upgrade by setting the `KUBECONFIG` environment variable, or using the `--kubeconfig` flag, in accordance with Kubernetes conventions[1290].
3. Upgrade the core addons in a cluster using the `dkp upgrade addons` command specifying the cluster infrastructure (choose `aws`, `azure`, `vsphere`, `vcd`, `eks`, `gcp`, `preprovisioned`) and the name of the cluster.

1289 https://cluster-api.sigs.k8s.io/
1290 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

If you need to verify or discover your cluster name to use with this example, first run the `kubectl get clusters` command.

Examples for upgrade core addons commands:

```
export CLUSTER_NAME=my-azure-cluster
dkp upgrade addons azure --cluster-name=${CLUSTER_NAME}
```

OR

```
export CLUSTER_NAME=my-aws-cluster
dkp upgrade addons aws --cluster-name=${CLUSTER_NAME}
```

The output for the AWS example should be similar to:

```
Generating addon resources
clusterresourceset.addons.cluster.x-k8s.io/calico-cni-installation-my-aws-cluster
upgraded
configmap/calico-cni-installation-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/tigera-operator-my-aws-cluster upgraded
configmap/tigera-operator-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/aws-ebs-csi-my-aws-cluster upgraded
configmap/aws-ebs-csi-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-my-aws-cluster upgraded
configmap/cluster-autoscaler-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-my-aws-cluster
upgraded
configmap/node-feature-discovery-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-my-aws-cluster
upgraded
configmap/nvidia-feature-discovery-my-aws-cluster upgraded
```

10.5.11.1 See Also:

DKP upgrade addons (see page 1935) for more CLI command help.

10.5.11.2 Next Step:

Enterprise: Upgrade the Management Cluster Kubernetes Version (see page 1735)

10.5.12 Enterprise: Upgrade the Management Cluster Kubernetes Version

Upgrade the Kubernetes version on the **Management cluster**.

Enterprise Gov Advanced

10.5.12.1 Upgrade the Kubernetes Version

When upgrading the Kubernetes version of a cluster:

1. Upgrade the control plane first using the infrastructure specific command.
    a. NOTE the additional considerations for FIPS if using FIPS configuration.
2. Upgrade the node pools second using the infrastructure specific command.
    a. NOTE the additional considerations for FIPS if using FIPS configuration.
3. If you have any additional managed, you need to upgrade the core addons and Kubernetes version for each one. Attached clusters would need the Kubernetes version upgraded to a supported DKP version using the tool that created that cluster.
4. During an upgrade, you would need to create new AMI's or images with the K8S version to which you want to upgrade which requires selecting your OS and ensuring you have the current supported version (see page 67). Build a new image if applicable.
    - If an AMI was specified when initially creating a cluster for AWS, you must build a new one with Konvoy Image Builder (see page 1634) and set the flag(s) in the update commands. Either AMI ID `--ami AMI_ID`, or the lookup image flags: `--ami-owner AWS_ACCOUNT_ID`, `--ami-base-os ubuntu-20.04`, and `--ami-format 'example-{{.BaseOS}}-?{{.K8sVersion}}-*'`.

      ⚠ The AMI lookup method will return an error if the lookup uses the upstream CAPA account ID.
    - If an Azure Machine Image was specified for Azure, you must build a new one with Konvoy Image Builder (see page 1642).
    - If a vSphere template Image was specified for vSphere, you must build a new one with Konvoy Image Builder (see page 1652).
    - You must build a new GCP image with Konvoy Image Builder (see page 1490).
5. Upgrade the Kubernetes version of the control plane. Each cloud provider has distinctive commands. Below is the AWS command example. Select the drop-down menu next to your provider for compliant CLI.
   NOTE: The first example below is for AWS. If you created your initial cluster with a custom AMI using the `--ami` flag, it is required to set the `--ami` flag during the Kubernetes upgrade.

```
dkp update controlplane aws --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7
```

Azure

```
dkp update controlplane azure --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --compute-gallery-id <Azure Compute Gallery built by KIB for
 Kubernetes v1.28.7>
```

- If these fields were specified in the override file (see page 1670) during image creation (see page 1642), the flags must be used in upgrade:
  - `--plan-offer`, `--plan-publisher` and `--plan-sku`
  - ```
    --plan-offer rockylinux-9
    --plan-publisher erockyenterprisesoftwarefoundationinc1653071250513
    --plan-sku rockylinux-9
    ```

vSphere

```
dkp update controlplane vsphere --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --vm-template <vSphere template built by KIB for Kubernetes v1.28.7>
```

VCD

```
dkp update controlplane vcd --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --catalog <tenant catalog image> --vapp-template <vApp template built
in vSphere KIB for Kubernetes v1.28.7>
```

GCP

```
dkp update controlplane gcp --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --image=projects/${GCP_PROJECT}/global/images/<GCP image built by KIB
for Kubernetes v1.28.7>
```

Pre-provisioned

```
dkp update controlplane preprovisioned --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7
```

EKS

```
dkp update controlplane eks --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.26.12
```

Additional Considerations for upgrading a FIPS cluster:
```

If upgrading a FIPS cluster, to correctly upgrade the Kubernetes version, instead run the command shown below which includes the etcd FIPS information as well:

```
dkp update controlplane aws --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7+fips.0 --ami=<ami-with-fips-id>
```

The output should be similar to the below example, with the provider name corresponding to the CLI you executed from the choices above:

```
Updating control plane resource controlplane.cluster.x-k8s.io/v1beta1,
Kind=KubeadmControlPlane default/my-aws-cluster-control-plane
Waiting for control plane update to finish.
 ✓ Updating the control plane
```

Some advanced options are available for various providers. To see all the options for your particular provider, run this command `dkp update controlplane aws|vsphere|preprovisioned|azure|gcp|eks --help` for more advance options like the example below:

This example for AWS AMI instance type: `aws: --ami, --instance-type` would be some of the options mentioned in the note above.

NOTE: The command `dkp update controlplane {provider}` has a 30 minute default timeout for the update process to finish. If you see the error " `timed out waiting for the condition` ", you can check the control plane nodes version using the command `kubectl get machines -o wide $KUBECONFIG` before trying again.

6. Upgrade the Kubernetes version of your node pools. Upgrading a nodepool involves draining the existing nodes in the nodepool and replacing them with new nodes. In order to ensure minimum downtime and maintain high availability of the critical application workloads during the upgrade process, we recommend deploying Pod Disruption Budget (Disruptions[1291]) for your critical applications. For more information, refer to Update Cluster Nodepools (see page 1620) documentation.

a. First, get a list of all node pools available in your cluster by running the following command:

```
dkp get nodepool --cluster-name ${CLUSTER_NAME}
```

b. Select the nodepool you want to upgrade with the command below:

1291 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/

```
export NODEPOOL_NAME=my-nodepool
```

c. Then update the selected nodepool using the command below. The first example command shows AWS language, so select the drop-down menu for your provider for the correct command. Execute the `update` command for each of the node pools listed in the previous command:

NOTE: The first example below is for AWS. If you created your initial cluster with a custom AMI using the `--ami` flag, it is required to set the `--ami` flag during the Kubernetes upgrade.

```
dkp update nodepool aws ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7
```

Azure

```
dkp update nodepool azure ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.28.7 --compute-gallery-id <Azure Compute Gallery built by KIB
for Kubernetes v1.28.7>
```

- If these fields were specified in the override file (see page 1670) during image creation (see page 1642), the flags must be used in upgrade:
  - `--plan-offer`, `--plan-publisher` and `--plan-sku`
  - ```
    --plan-offer rockylinux-9
    --plan-publisher erockyenterprisesoftwarefoundationinc1653071250513
    --plan-sku rockylinux-9
    ```

vSphere

```
dkp update nodepool vsphere ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.28.7 --vm-template <vSphere template built by KIB for
 Kubernetes v1.28.7>
```

VCD

```
dkp update nodepool vcd ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --catalog <tenant catalog image> --vapp-template <vApp template built
in vSphere KIB for Kubernetes v1.28.7>
```

GCP

```
dkp update nodepool gcp ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --image=projects/${GCP_PROJECT}/global/images/<GCP image built by KIB
for Kubernetes v1.28.7>
```
```

Pre-provisioned

```
dkp update nodepool preprovisioned ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.28.7
```

EKS

```
dkp update nodepool eks ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.26.12
```

The output should be similar to the following, with the name of the infrastructure provider shown accordingly:

```
Updating node pool resource cluster.x-k8s.io/v1beta1, Kind=MachineDeployment default/
my-aws-cluster-my-nodepool
Waiting for node pool update to finish.
 ✓ Updating the my-aws-cluster-my-nodepool node pool
```

d. Repeat this step for each additional node pool.

Additional Considerations for upgrading a FIPS cluster:

If upgrading a FIPS cluster, to correctly upgrade the Kubernetes version, instead run the command shown below:

```
dkp update nodepool aws ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7+fips.0 --ami=<ami-with-fips-id>
```

When all nodepools have been updated, your upgrade is complete. For the overall process for upgrading to the latest version of DKP, refer back to Upgrade DKP (see page 1705) for more details.

10.5.12.2 Next Step:

Enterprise: Upgrade Managed Clusters (see page 1739)

10.5.13 Enterprise: Upgrade Managed Clusters

10.5.13.1 Upgrade Managed Clusters

Enterprise Gov Advanced

If you have managed clusters, follow these steps to upgrade each cluster similar to the management cluster process order:

1. Using the kubeconfig of your management cluster, find your cluster name and be sure to copy the information for all of your clusters:

```
kubectl get clusters -A
```

2. Set your cluster variable:

```
export CLUSTER_NAME=<your-managed-cluster-name>
```

3. Set your cluster's workspace variable:

```
export CLUSTER_WORKSPACE=<your-workspace-namespace>
```

4. Then, upgrade the core addons (replacing `aws` with whatever infrastructure provider you would be using):

```
dkp upgrade addons aws --cluster-name=${CLUSTER_NAME} -n ${CLUSTER_WORKSPACE}
```

10.5.13.2 Upgrade Kubernetes Version on a Managed Cluster

After you complete the previous steps for all managed clusters and you update your core addons, begin upgrading the Kubernetes version.

You should first complete the upgrade of your Kommander Management Cluster before upgrading any managed clusters.

1. Use this command to begin upgrading the Kubernetes version:

```
dkp update controlplane aws --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 -n ${CLUSTER_WORKSPACE}
```

For EKS

```
dkp update controlplane eks --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.26.12 -n ${CLUSTER_WORKSPACE}
```

For Azure

```
dkp update controlplane azure --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --compute-gallery-id <Azure Compute Gallery built by KIB for
 Kubernetes v1.28.7> -n ${CLUSTER_WORKSPACE}
```

For vSphere

```
dkp update controlplane vsphere --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --vm-template <vSphere template built by KIB for Kubernetes v1.28.7>
-n ${CLUSTER_WORKSPACE}
```

For VCD

```
dkp update controlplane vcd --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --catalog <tenant catalog image> --vapp-template <vApp template built
in vSphere KIB for Kubernetes v1.28.7> -n ${CLUSTER_WORKSPACE}
```

For GCP

```
dkp update controlplane gcp --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --image=projects/${GCP_PROJECT}/global/images/<GCP image built by KIB
for Kubernetes v1.28.7> -n ${CLUSTER_WORKSPACE}
```

For Pre-provisioned

```
dkp update controlplane preprovisioned --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 -n ${CLUSTER_WORKSPACE}
```

2. Get a list of all node pools available in your cluster by running the following command:

```
dkp get nodepools -c ${CLUSTER_NAME} -n ${CLUSTER_WORKSPACE}

export NODEPOOL_NAME=<my-nodepool>
```

3. Use this command to upgrade the node pools:

```
dkp update nodepool aws ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 -n ${CLUSTER_WORKSPACE}
```

For EKS

```
dkp update nodepool eks ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.26.12 -n ${CLUSTER_WORKSPACE}
```

For Azure

```
dkp update nodepool azure ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.28.7 --compute-gallery-id <Azure Compute Gallery built by KIB
for Kubernetes v1.28.7> -n ${CLUSTER_WORKSPACE}
```

For vSphere

```
dkp update nodepool vsphere ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.28.7 --vm-template <vSphere template built by KIB for
 Kubernetes v1.28.7> -n ${CLUSTER_WORKSPACE}
```

For VCD

```
dkp update nodepool vcd ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --catalog <tenant catalog image> --vapp-template <vApp template built
in vSphere KIB for Kubernetes v1.28.7>  -n ${CLUSTER_WORKSPACE}
```

For GCP

```
dkp update nodepool gcp ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --image=projects/${GCP_PROJECT}/global/images/<GCP image built by KIB
for Kubernetes v1.28.7> -n ${CLUSTER_WORKSPACE}
```

For Pre-provisioned

```
dkp update nodepool preprovisioned ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.28.7 -n ${CLUSTER_WORKSPACE}
```

10.5.13.3 Upgrade Attached Cluster

Since any attached clusters you have are managed by their corresponding cloud provider, none of the components would be upgraded using the DKP process. The tool used to create the cluster is used to upgrade it. Ensure that it has the Kubernetes version that is compatible with the versions we support.

10.5.14 Enterprise: Upgrade images used by Catalog Applications

To reduce the risk caused by vulnerabilities present in older Kafka and Zookeeper images, you should upgrade image references in the `ZookeeperCluster` and `KafkaCluster` custom resources on attached clusters.

Please note that you must perform this step after completing your upgrade of DKP. You should perform this step **after** upgrading the `zookeeper-operator` and `kafka-operator` applications themselves.

Note that these images can be upgraded individually, on a cluster-by-cluster basis.

10.5.14.1 Upgrading a Zookeeper image reference in a `ZookeeperCluster` resource

The DKP catalog in 2.7 includes an updated Zookeeper 3.7.2 image, which does not contain CVEs present in earlier versions. This image is hosted at `ghcr.io/mesosphere/zookeeper:0.2.15-d2iq`. To update an existing Zookeeper cluster to use this image, perform the following steps:

1. Ensure that your workloads that use this Zookeeper cluster are compatible with Zookeeper 3.7.2
2. Set the `spec.image.repository` and `spec.image.tag` fields of each `ZookeeperCluster` custom resource to reference the new image.
   For example, you can use `kubectl patch` to upgrade the image in a `ZookeeperCluster` named `zk-3` that has been manually created in a namespace `ws-1` as follows:

   ```
   kubectl patch zookeepercluster -n ws-1 zk-3 --type='json' -p='[{"op":
   "replace", "path": "/spec/image/repository", "value": "ghcr.io/mesosphere/
   zookeeper"}, {"op": "replace", "path": "/spec/image/tag", "value": "0.2.15-
   d2iq"}]'
   ```

For more information, refer to Zookeeper Operator documentation: https://github.com/pravega/zookeeper-operator/blob/master/README.md#trigger-the-upgrade-manually

10.5.14.2 Upgrading the Kafka image reference in a `KafkaCluster` resource

The DKP catalog in 2.7 includes an updated Kafka 3.4.1 image, which does not contain CVEs present in earlier versions. This image is hosted at `ghcr.io/banzaicloud/kafka:2.13-3.4.1`. To update an existing `KafkaCluster` to use this image, perform the following steps:

1. Ensure that all workloads that use this Kafka cluster are compatible with this version of Kafka.
2. Identify the current version of Kafka, which is specified in the `.spec.clusterImage` field of the `KafkaCluster` resource.
3. Ensure that the `spec.readOnlyConfig` field of the `KafkaCluster` contains a `inter.broker.protocol.version=X.Y.Z` line, where `X.Y.Z` stands for the current version of Kafka, and no other lines setting this value are present in the `KafkaCluster` resource. This ensures all the brokers in the cluster can communicate with each other as the cluster is upgraded using the specified protocol version.

   For example, if you use the `kubectl edit` command to perform this step on a `KafkaCluster`

currently using the `ghcr.io/banzaicloud/kafka:2.13-2.8.1` image, the resulting KafkaCluster manifest should resemble this:

```
apiVersion: kafka.banzaicloud.io/v1beta1
...
spec:
  clusterImage: "ghcr.io/banzaicloud/kafka:2.13-2.8.1"
  ...
  readOnlyConfig: |
    auto.create.topics.enable=false
    cruise.control.metrics.topic.auto.create=true
    cruise.control.metrics.topic.num.partitions=1
    cruise.control.metrics.topic.replication.factor=2
    inter.broker.protocol.version=2.8.1
    ...
```

For more information, refer to Kafka upgrade documentation: https://kafka.apache.org/documentation/#upgrade

4. Wait until `kafka-operator` reconciles the broker protocol change. `kubectl get -A kafkaclusters` should report `ClusterRunning` status for this KafkaCluster.
5. Set the `spec.clusterImage` to the chosen image.
   In the example above, the `KafkaCluster` manifest will look like this:

```
...
spec:
  clusterImage: "ghcr.io/banzaicloud/kafka:2.13-3.4.1"
  ...
  readOnlyConfig: |
    ...
    inter.broker.protocol.version=2.8.1
    ...
```

6. Wait until `kafka-operator` fully applies the image change. After that, the cluster will run a new version of Kafka, but will still use the old protocol version.
7. Verify the behavior and performance of this Kafka cluster and workloads that use it.
   WARNING: This verification should not be postponed, because the next step can make a rollback to the old version of Kafka impossible.
8. Bump the protocol version to the current one by modifying the `spec.readOnlyConfig` field so that `inter.broker.protocol.version` equals the new Kafka version.
   In the example above, the `KafkaCluster` manifest will look like this:

```
...
```

```
spec:
  clusterImage: "http://ghcr.io/banzaicloud/kafka:2.13-3.4.1"
  ...
  readOnlyConfig: |
    ...
    inter.broker.protocol.version=3.4.1
    ...
```

9. Wait until `kafka-operator` fully applies the protocol version change. The upgrade of the Kafka image for this cluster is now complete.

10.6 Upgrade DKP Essential

10.6.1 Steps to upgrade DKP via CLI

The DKP upgrade represents an important step of your environment's lifecycle, as it ensures that you are up-to-date with the latest features and can benefit from the most recent improvements, enhanced cluster management, and better performance. If you have are on this scenario of the upgrade section of documentation, you have an Essential License. If you have an Enterprise license (see page 85), please jump to that section of the Upgrade DKP Enterprise (see page 1716) portion of documentation.

Ensure that all platform applications in the management cluster have been upgraded to avoid compatibility issues with the Kubernetes version (see page 1706) included in this release. This is done automatically when upgrading Kommander, so ensure that you follow these sections in order to upgrade the Kommander component prior to upgrading the Konvoy component.

If you have more than one Essential cluster, repeat all of these steps for each Essential cluster. For a full list of DKP Essential features, see DKP Essential (see page 91).

Before you begin the upgrade, ensure you review the Upgrade Prerequisites (see page 1709).

10.6.2 Overview of Steps in Order

This section describes how to upgrade your DKP clusters to the latest DKP version.

- Essential: For Air-gapped Environments Only (see page 1746)
- Essential: Upgrade the Essential Cluster and Platform Applications (see page 1748)
- Essential: Upgrade the Cluster CAPI Components (see page 1750)
- Essential: Upgrade the Cluster Core Addons (see page 1751)
- Essential Upgrade Kubernetes Version (see page 1753)

10.6.3 Next Step:

If you are in an air-gapped environment, be sure to select that choice to find out how to load your local registry first. Otherwise, your next step will be Essential: Upgrade the Essential Cluster and Platform Applications (see page 1748).

- Essential: For Air-gapped Environments Only (see page 1746)

10.6.4 Essential: For Air-gapped Environments Only

If you are operating in an air-gapped environment, a local container registry (see page 1606) containing all the necessary installation images, including the Kommander images is required. See below for how to push the necessary images to this registry.

- Download (see page 76) the Complete DKP Air-gapped Bundle for this release (i.e. `dkp-air-gapped-bundle_v2.7.0_linux_amd64.tar.gz`)
- Connectivity:
  - Your Essential cluster must be able to connect to the local registry.

10.6.4.1 Extract Air-gapped Images and Set Variables

Follow these steps to **extract** the air-gapped image bundles into your private registry:

1. Assuming you have downloaded `dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz`, extract the tarball to a local directory:

   ```
   tar -xzvf dkp-air-gapped-bundle_v2.8.1_linux_amd64.tar.gz
   ```

2. The directory structure after extraction can be accessed in subsequent steps using commands to access files from different directories. EX: For the bootstrap, change your directory to the `dkp-<version>` directory similar to example below depending on your current location:

   ```
   cd dkp-v2.8.1
   ```

3. Set an environment variable with your registry address:

   ```
   export REGISTRY_URL="<https/http>://<registry-address>:<registry-port>"
   export REGISTRY_USERNAME=<username>
   export REGISTRY_PASSWORD=<password>
   ```

10.6.4.1.1 Load Images to your Private Registry - Kommander

Load Kommander images to your Private Registry

For the **air-gapped** `kommander` image bundle, run the command below:

Run the following command to load the image bundle:

```
dkp push bundle --bundle ./container-images/kommander-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

10.6.4.1.2 Load Images to your Private Registry - Konvoy

Before creating or upgrading a Kubernetes cluster, you need to load the required images in a local registry if operating in an air-gapped environment. This registry must be accessible from both the bastion machine (see page 1068) and either the AWS EC2 instances or other machines that will be created for the Kubernetes cluster.

If you do not already have a local registry set up, refer to Local Registry Tools (see page 1606) page for more information.

Execute the following command to load the air-gapped image bundle into your private registry:

```
dkp push bundle --bundle ./container-images/konvoy-image-bundle-v2.8.1.tar --to-
registry=${REGISTRY_URL} --to-registry-username=${REGISTRY_USERNAME} --to-registry-
password=${REGISTRY_PASSWORD}
```

It may take some time to push all the images to your image registry, depending on the performance of the network between the machine you are running the script on and the registry.

10.6.4.1.3 Next Step:

Essential: Upgrade the Essential Cluster and Platform Applications (see page 1748)

10.6.5 Essential: Upgrade the Essential Cluster and Platform Applications

This section describes how to upgrade your Kommander Management cluster and all Platform Applications to their supported versions in air-gapped, non-air-gapped and on-prem environments. To prevent compatibility issues, you must first upgrade the Kommander component on your Essential Cluster (see page 81) before upgrading to DKP.

It is important you upgrade Kommander BEFORE upgrading the Kubernetes version. This ensures that any changes required for new or changed Kubernetes API's are already present.

10.6.5.1 Upgrade Kommander

10.6.5.1.1 Prerequisite

- Use the `--kubeconfig=${CLUSTER_NAME}.conf` flag or set the `KUBECONFIG` environment variable to ensure that you **upgrade Kommander on the right cluster**. For alternatives and recommendations around setting your context, refer to Provide Context for Commands with a kubeconfig File (see page 106).
- If you have DKP Insights installed, ensure you **uninstall** it before upgrading DKP. See DKP Insights Upgrade from DKP 2.5.x to 2.6.0[1292] for more information.

Limited availability of DKP resources

- If you have configured a **custom domain**, running the `upgrade` command could result in an inaccessibility of your services via your custom domain for a few minutes.
- The **DKP UI** and other APIs may be inconsistent or unavailable until the upgrade is complete.

10.6.5.1.2 Upgrade

Use the DKP CLI to upgrade Kommander and all the Platform Applications in the Management Cluster:

For air-gapped environments:

```
dkp upgrade kommander \
--charts-bundle ./application-charts/dkp-kommander-charts-bundle-v2.7.0.tar.gz \
```

1292 https://d2iq.atlassian.net/wiki/spaces/DINS/pages/247169025

```
--kommander-applications-repository ./application-repositories/kommander-
applications-v2.7.0.tar.gz
```

The output looks similar to this:

```
✓ Ensuring upgrading conditions are met
✓ Ensuring application definitions are updated
✓ Ensuring helm-mirror implementation is migrated to ChartMuseum
...
```

For non-air-gapped environments:

```
dkp upgrade kommander
```

If you want to disable the AI Navigator, add the flag `--disable-appdeployments ai-navigator-app` to this command.

The output looks similar to this:

```
✓ Ensuring upgrading conditions are met
✓ Ensuring application definitions are updated
✓ Ensuring helm-mirror implementation is migrated to ChartMuseum
...
```

10.6.5.1.3 Troubleshooting

1. If the upgrade fails, run the following command to get more information on the upgrade process:

   ```
   dkp upgrade kommander -v 6
   ```

2. If you find any `HelmReleases` in a “broken” release state such as “exhausted” or “another rollback/release in progress”, you can trigger a reconciliation of the `HelmRelease` using the following commands:

   ```
   kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"
   op": "replace", "path": "/spec/suspend", "value": true}]'
   ```

```
kubectl -n kommander patch helmrelease <HELMRELEASE_NAME> --type='json' -p='[{"
op": "replace", "path": "/spec/suspend", "value": false}]'
```

10.6.5.1.4 Next Step:

Essential: Upgrade the Cluster CAPI Components (see page 1750)

10.6.6 Essential: Upgrade the Cluster CAPI Components

Upgrade the CAPI Components on the Management cluster.

For a Pre-provisioned air-gapped environment only, ensure you have uploaded the artifacts (see page 1712).

10.6.6.1 Upgrade the CAPI Components

New versions of DKP come pre-bundled with newer versions of CAPI, newer versions of infrastructure providers, or new infrastructure providers. When using a new version of the DKP CLI, upgrade all of these components first.

If you are running on more than one management cluster, you must upgrade the CAPI components on each of these clusters.

Ensure your `dkp` configuration references the management cluster where you want to run the upgrade by setting the `KUBECONFIG` environment variable, or using the `--kubeconfig` flag, in accordance with Kubernetes conventions[1293].

Execute the upgrade command for the CAPI components.

If you created CAPI components using flags to specify values, use those same flags during **Upgrade** to preserve existing values while setting additional values.

- Refer to dkp create cluster aws (see page 1856) for flag descriptions for `--with-aws-bootstrap-credentials` and `--aws-service-endpoints`

1293 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

- Refer to the HTTP section for details: Create CAPI Components with HTTP/HTTPS Proxy Settings (see page 1055)

```
dkp upgrade capi-components
```

The output resembles the following:

```
✓ Upgrading CAPI components
✓ Waiting for CAPI components to be upgraded
✓ Initializing new CAPI components
```

If the upgrade fails, review the prerequisites section and ensure that you've followed the steps in the DKP Upgrade (see page 1705) overview. Furthermore, ensure you have adhered to the Prerequisites at the top of this page.

10.6.6.2 Next Step:

Essential: Upgrade the Cluster Core Addons (see page 1751)

10.6.7 Essential: Upgrade the Cluster Core Addons

For Pre-provisioned air-gapped environments only, you must run `konvoy-image upload artifacts` to copy the artifacts (see page 1120) onto the cluster hosts **before** you begin the Upgrade the CAPI Components section below.

```
konvoy-image upload artifacts \
    --container-images-dir=./artifacts/images/ \
    --os-packages-bundle=./artifacts/$OS_PACKAGES_BUNDLE \
    --containerd-bundle=artifacts/$CONTAINERD_BUNDLE \
    --pip-packages-bundle=./artifacts/pip-packages.tar.gz
```

10.6.7.1 Upgrade the Core Addons

To install the core addons, DKP relies on the `ClusterResourceSet` Cluster API feature[1294]. In the CAPI component upgrade, we deleted the previous set of outdated global `ClusterResourceSets` because in past releases, some addons were installed using a global configuration. In order to support individual cluster upgrades, DKP now installs all addons with a unique set of `ClusterResourceSets` and corresponding referenced resources, all named using the cluster's name as a suffix. For example: `calico-cni-installation-my-aws-cluster`.

1294 https://cluster-api.sigs.k8s.io/

If you modify any of the `ClusterResourceSet` definitions, these changes are **not** be preserved when running the command `dkp upgrade addons`. You must use the `--dry-run -o yaml` options to save the new configuration to a file and continue the same changes upon each upgrade.

Your cluster comes preconfigured with a few different core addons that provide functionality to your cluster upon creation. These include: CSI, CNI, Cluster Autoscaler, and Node Feature Discovery. New versions of DKP may come pre-bundled with newer versions of these addons.

If you have more than one essential cluster, ensure your `dkp` configuration references the cluster where you want to run the upgrade by setting the `KUBECONFIG` environment variable, or using the `--kubeconfig` flag, in accordance with Kubernetes conventions[1295].

Perform the following steps to update your addons.

1. Ensure your `dkp` configuration references the cluster where you want to run the upgrade by setting the `KUBECONFIG` environment variable, or using the `--kubeconfig` flag, in accordance with Kubernetes conventions[1296].
2. Upgrade the core addons in a cluster using the `dkp upgrade addons` command specifying the cluster infrastructure (choose `aws`, `azure`, `vsphere`, `vcd`, `eks`, `gcp`, `preprovisioned`) and the name of the cluster.

Examples for upgrade core addons commands:

```
export CLUSTER_NAME=my-azure-cluster
dkp upgrade addons azure --cluster-name=${CLUSTER_NAME}
```

OR

```
export CLUSTER_NAME=my-aws-cluster
dkp upgrade addons aws --cluster-name=${CLUSTER_NAME}
```

The output for the AWS example should be similar to:

```
Generating addon resources
clusterresourceset.addons.cluster.x-k8s.io/calico-cni-installation-my-aws-cluster
upgraded
configmap/calico-cni-installation-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/tigera-operator-my-aws-cluster upgraded
configmap/tigera-operator-my-aws-cluster upgraded
```

1295 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/
1296 https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/

```
clusterresourceset.addons.cluster.x-k8s.io/aws-ebs-csi-my-aws-cluster upgraded
configmap/aws-ebs-csi-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/cluster-autoscaler-my-aws-cluster upgraded
configmap/cluster-autoscaler-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/node-feature-discovery-my-aws-cluster
upgraded
configmap/node-feature-discovery-my-aws-cluster upgraded
clusterresourceset.addons.cluster.x-k8s.io/nvidia-feature-discovery-my-aws-cluster
upgraded
configmap/nvidia-feature-discovery-my-aws-cluster upgraded
```

10.6.7.1.1 See Also

DKP upgrade addons (see page 1935) for more CLI command help.

10.6.7.2 Next Step:

Essential Upgrade Kubernetes Version (see page 1753)

10.6.8 Essential Upgrade Kubernetes Version

10.6.8.1 Upgrade the Kubernetes Version

When upgrading the Kubernetes version of a cluster:

1. Upgrade the control plane first using the infrastructure specific command.
    a. NOTE the additional considerations for FIPS if using FIPS configuration.
2. Upgrade the node pools second using the infrastructure specific command.
    a. NOTE the additional considerations for FIPS if using FIPS configuration.
3. Build a new image if applicable.
    - If an AMI was specified when initially creating a cluster for AWS, you must build a new one with Konvoy Image Builder (see page 1634) and set the flag(s) in the update commands. Either AMI ID `--ami AMI_ID`, or the lookup image flags: `--ami-owner AWS_ACCOUNT_ID`, `--ami-base-os ubuntu-20.04`, and `--ami-format 'example-{{.BaseOS}}-?{{.K8sVersion}}-*'`.

      ⚠ The AMI lookup method will return an error if the lookup uses the upstream CAPA account ID.
    - If an Azure Machine Image was specified for Azure, you must build a new one with Konvoy Image Builder (see page 1642).
    - If a vSphere template Image was specified for vSphere, you must build a new one with Konvoy Image Builder (see page 1652).
    - You must build a new GCP image with Konvoy Image Builder (see page 1490).

4. Upgrade the Kubernetes version of the control plane. Each cloud provider has distinctive commands. Below is the AWS command example. Select the drop-down menu next to your provider for compliant CLI.
   NOTE: The first example below is for AWS. If you created your initial cluster with a custom AMI using the `--ami` flag, it is required to set the `--ami` flag during the Kubernetes upgrade.

   ```
   dkp update controlplane aws --cluster-name=${CLUSTER_NAME} --kubernetes-
   version=v1.28.7
   ```

Azure

```
dkp update controlplane azure --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --compute-gallery-id <Azure Compute Gallery built by KIB for
 Kubernetes v1.28.7>
```

- If these fields were specified in the override file (see page 1670) during image creation (see page 1642), the flags must be used in upgrade:
  - `--plan-offer`, `--plan-publisher` and `--plan-sku`
  - ```
    --plan-offer rockylinux-9
    --plan-publisher erockyenterprisesoftwarefoundationinc1653071250513
    --plan-sku rockylinux-9
    ```

vSphere

```
dkp update controlplane vsphere --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --vm-template <vSphere template built by KIB for Kubernetes v1.28.7>
```

VCD

```
dkp update controlplane vcd --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --catalog <tenant catalog vApp template> --vapp-template <vApp
template built in vSphere KIB for Kubernetes v1.28.7>
```

GCP

```
dkp update controlplane gcp --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --image=projects/${GCP_PROJECT}/global/images/<GCP image built by KIB
for Kubernetes v1.28.7>
```

Pre-provisioned
```

```
dkp update controlplane preprovisioned --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7
```

EKS

```
dkp update controlplane eks --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.26.12
```

Additional Considerations for upgrading a FIPS cluster:

If upgrading a FIPS cluster, to correctly upgrade the Kubernetes version, instead run the command shown below which contains the etcd version as well:

```
dkp update controlplane aws --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7+fips.0 --ami=<ami-with-fips-id>
```

The output should be similar to the below example, with the provider name corresponding to the CLI you executed from the choices above:

```
Updating control plane resource controlplane.cluster.x-k8s.io/v1beta1,
Kind=KubeadmControlPlane default/my-aws-cluster-control-plane
Waiting for control plane update to finish.
 ✓ Updating the control plane
```

Some advanced options are available for various providers. To see all the options for your particular provider, run this command `dkp update controlplane aws|vsphere|preprovisioned|azure|gcp|eks --help` for more advance options like the example below:

This example for AWS AMI instance type: `aws: --ami, --instance-type` would be some of the options mentioned in the note above.

NOTE: The command `dkp update controlplane {provider}` has a 30 minute default timeout for the update process to finish. If you see the error "`timed out waiting for the condition`", you can check the control plane nodes version using the command `kubectl get machines -o wide --kubeconfig $KUBECONFIG` before trying again.

5. Upgrade the Kubernetes version of your node pools. Upgrading a nodepool involves draining the existing nodes in the nodepool and replacing them with new nodes. In order to ensure minimum downtime and maintain high availability of the critical application workloads during the upgrade process, we recommend

deploying Pod Disruption Budget (Disruptions[1297]) for your critical applications. For more information, refer to Update Cluster Nodepools[1298] documentation.

a. First, get a list of all node pools available in your cluster by running the following command:

```
dkp get nodepool --cluster-name ${CLUSTER_NAME}
```

b. Select the nodepool you want to upgrade with the command below:

```
export NODEPOOL_NAME=my-nodepool
```

c. Then update the selected nodepool using the command below. Upgrading a node pool involves draining the existing nodes in the node pool and replacing them with new nodes. we recommend deploying Pod Disruption Budget (Disruptions[1299]) for your critical applications. Refer to Update Cluster Nodepools (see page 1620) for more information.

The first example command shows AWS language, so select the drop-down menu for your provider for the correct command. Execute the `update` command for each of the node pools listed in the previous command:

NOTE: The first example below is for AWS. If you created your initial cluster with a custom AMI using the `--ami` flag, it is required to set the `--ami` flag during the Kubernetes upgrade.

```
dkp update nodepool aws ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7
```

Azure

```
dkp update nodepool azure ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.28.7 --compute-gallery-id <Azure Compute Gallery built by KIB
for Kubernetes v1.28.7>
```

- If these fields were specified in the override file (see page 1670) during image creation (see page 1642), the flags must be used in upgrade:
  - `--plan-offer`, `--plan-publisher` and `--plan-sku`
  - ```
    --plan-offer rockylinux-9
    --plan-publisher erockyenterprisesoftwarefoundationinc1653071250513
    --plan-sku rockylinux-9
    ```

vSphere

1297 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/
1298 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/213191226
1299 https://kubernetes.io/docs/concepts/workloads/pods/disruptions/
    ```

```
dkp update nodepool vsphere ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.28.7 --vm-template <vSphere template built by KIB for
 Kubernetes v1.28.7>
```

VCD

```
dkp update nodepool vcd ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --catalog <tenant catalog vApp template> --vapp-template <vApp
template built in vSphere KIB for Kubernetes v1.28.7>
```

GCP

```
dkp update nodepool gcp ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7 --image=projects/${GCP_PROJECT}/global/images/<GCP image built by KIB
for Kubernetes v1.28.7>
```

Pre-provisioned

```
dkp update nodepool preprovisioned ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --
kubernetes-version=v1.28.7
```

EKS

```
dkp update nodepool eks ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.26.12
```

Additional Considerations for upgrading a FIPS cluster:

If upgrading a FIPS cluster, to correctly upgrade the Kubernetes version, instead run the command shown below:

```
dkp update nodepool aws ${NODEPOOL_NAME} --cluster-name=${CLUSTER_NAME} --kubernetes-
version=v1.28.7+fips.0 --ami=<ami-with-fips-id>
```

The output should be similar to the following, with the name of the infrastructure provider shown accordingly:

```
Updating node pool resource cluster.x-k8s.io/v1beta1, Kind=MachineDeployment default/
my-aws-cluster-my-nodepool
Waiting for node pool update to finish.
 ✓ Updating the my-aws-cluster-my-nodepool node pool
```

d. Repeat this step for each additional node pool.

When all nodepools have been updated, your upgrade is complete. For the overall process for upgrading to the latest version of DKP, refer back to Upgrade DKP (see page 1705) for more details.

11 Troubleshooting Guide

These five sections represent the most common areas of questions directed at our Support Department. By starting here, you can answer those frequently asked questions on your own timetable.

- Gather Environment Data for D2iQ Support (see page 1758)
- Common Information Discovery Commands (see page 1767)
- Application Troubleshooting (see page 1771)
- Application YAML Customizations (see page 1777)
- Object Locations and Explanations (see page 1786)

11.1 Gather Environment Data for D2iQ Support

The following pages provide the steps to troubleshoot a Kubernetes cluster customization. These subtopics provide a way to supply a support bundle file back to D2iQ for further analysis.

- Generate a Support Bundle (see page 1758)
- SSH and Ansible (see page 1761)
- Custom Collectors (see page 1763)

11.1.1 Generate a Support Bundle

Follow these instructions to generate a support bundle with data collected for the last 48 hours of the life of the cluster.

11.1.1.1 Prerequisites

Before generating a support bundle, verify that you have:

- An AMD64-based Linux or macOS machine with a supported version of the operating system.
- A running Kubernetes cluster.
- Access to the DKP CLI (see page 1835).

11.1.1.1.1 Diagnostic Bundle

`dkp diagnose` was developed by D2iQ and builds on the open source `troubleshoot.sh` project.

The command `dkp diagnose` is based on version `0.13.16` of `troubleshoot.sh` with custom modifications. The D2iQ fork is open source and available from this public GitHub repository[1300].

`dkp diagnose` supports multiple support bundle collectors[1301] and can be configured as a `SupportBundle` Kubernetes resource in a YAML file.

The following list is the minimum set of resources that is required to debug a cluster, but can be further customized.

The bundle uses the following collectors:

- clusterInfo[1302] collects basic information about the cluster
- clusterResources[1303] collects a subset of available resources in the cluster
- configMap[1304] collects the values of Kubernetes ConfigMaps
- secrets[1305] collects the names and metadata of secrets, but NOT the secret values or keys
- execCopyFromHost (see page 0) runs a container on each node on the cluster and copies the created data
- allLogs (see page 1763) is capable of collecting logs from all containers on the cluster
- cpuMemoryUtilisation collects the average CPU and memory utilization of the last 30 days.

11.1.1.1.2 Generate a Support Bundle

The command `dkp diagnose` uses the same Kubernetes configuration as `kubectl`. `dkp diagnose` can also be pointed at a specific configuration by using the `--kubeconfig` parameter.

To generate the support bundle, perform the following steps:

1. Run the `dkp diagnose` command by running the default collectors configuration.

```
dkp diagnose
```

1300 https://github.com/mesosphere/troubleshoot
1301 https://troubleshoot.sh/docs/collect/all/
1302 https://troubleshoot.sh/docs/collect/cluster-info/
1303 https://troubleshoot.sh/docs/collect/cluster-resources/
1304 https://troubleshoot.sh/docs/collect/configmap/
1305 https://troubleshoot.sh/docs/collect/secret/

The output looks similar to this:

```
Collecting support bundle ...

support-bundle-2021-08-13T14_44_23.tar.gz
```

2. To view the bundle contents, extract the bundle (replacing `support-bundle-2021-08-13T14_44_23.tar.gz` with the location from the previous step):

```
tar -xzvf support-bundle-2021-08-13T14_44_23.tar.gz
```

3. A new directory named `support-bundle-<date-created>` is created. This directory contains the files specified:

```
ls support-bundle-2021-08-13T14_44_23
```

The output looks similar to this:

```
cluster-info  cluster-resources  configmaps  node-diagnostics  pod-logs
secrets  version.yaml
```

11.1.1.1.3 Collect Information from a Bootstrap Cluster

In the case where your bootstrap cluster has not yet pivoted towards your Konvoy cluster, you can collect log information from that bootstrap cluster as well, and there are a preconfigured set of relevant collectors. Specify an additional bootstrap cluster kubeconfig using the `--bootstrap-kubeconfig` parameter to activate bootstrap cluster diagnostics. You will receive an additional support bundle named `bootstrap-support-bundle-<date created>`.

Note that the bootstrap cluster diagnostics are independent of the configuration of the “main” or Konvoy cluster diagnostics. We run a static collector set that collects the following bootstrap cluster information:

- ClusterInfo
- ClusterResources
- AllLogs
- ConfigMaps
- Secrets

1. Run the `dkp diagnose` command with bootstrap bundle configuration.

```
dkp diagnose bundle.yaml
```

11.1.1.1.3.1 Customizations

To print the default collectors configuration, run the following command:

```
dkp diagnose default-config > bundle.yaml
```

Edit the file to make appropriate modifications.

> By default, `dkp diagnose` does not require that you supply a configuration. You can print the default bundle by running `dkp diagnose default-config`.

11.1.1.1.4 Next Topic

SSH and Ansible (see page 1761)

11.1.2 SSH and Ansible

11.1.2.1 SSH Fallback

In some cases the Kubernetes API is not available for the cluster. In those cases you can collect node level information using SSH access to the diagnosed nodes. Be aware that not all clusters have SSH access configured. If they do not then access using SSH fallback is not possible.

To get node level information from your cluster using SSH access, perform the following steps:

1. Enter the following command:

```
dkp diagnose ssh <path/to/ansible-inventory.yaml>
```

The `ansible-inventory.yaml` file specifies the nodes to access for data collection.

> This collector does not use the full Ansible `inventory.yaml` format only a limited subset to describe the infrastructure.

Only the following attributes of the `ansible-inventory.yaml` are supported. All other group definitions are ignored.

- Support for `all` shared variables.
- Support for `hosts` key in `all` groups.
- Supported behavioral inventory is limited to:
  - `ansible_host`
  - `ansible_port`
  - `ansible_user`
  - `ansible_ssh_private_key_file`

  The following is an example `inventory.yaml` file:

```
all:
  vars:
    ansible_user: centos
  hosts:
    host-1:
      ansible_host: 192.168.10.1
    host-2:
      ansible_host: 192.168.10.22
      ansible_port: 2222
```

More information on these Ansible parameters can be found in the Ansible user guide[1306].

All other group definitions in the `inventory.yaml` file are ignored.

Refer to the following example file:

```
all:
  vars:
    ansible_user: centos
  hosts:
    host-1:
      ansible_host: 192.168.10.1
    host-2:
      ansible_host: 192.168.10.22
      ansible_port: 2222
```

The fallback collector runs a bash script over SSH and copies the collected data. The format of the created bundle matches that of `dkp diagnose` collector generated bundles.

1306 https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#connecting-to-hosts-behavioral-inventory-parameters

```
node-diagnostics/<HOSTNAME_PORT>/data/
    - dmesg
    - ....
```

Redactors are supported and are in the same format as the main `dkp diagnose` command. Per node collection timeouts are supported using the `--timeout` parameter.

See also: dkp-cli (see page 1835)

11.1.2.2 Next Topic

Custom Collectors (see page 1763)

11.1.3 Custom Collectors

For creating diagnostic bundles, D2iQ is using a customized version of `troubleshoot.sh` integrated into `dkp-diagnose`.

11.1.3.1 Customizations

To meet the specific needs of diagnosing DKP 2 clusters we have developed custom collectors and modified the behavior of upstream collectors. Go to our repository for more information on the details[1307] of the changes.

11.1.3.1.1 `ExecCopyFromHost` Collector

This is a new collector created specifically for gathering host level information from cluster nodes. The collector allows you to run a provided container image in a privileged mode, as a root user, with additional Linux capabilities and with the host filesystem mounted in the container.

You can collect host level information other than copying host level files. (This is already possible with the `CopyFromHost` collector.) Like the `CopyFromHost` collector, this collector runs as a Kubernetes `DaemonSet` executed on all nodes in the system. The data produced by the container are copied from a pre-defined directory into the diagnostics bundle under each node name. The name of the parent directory, in the diagnostics bundle, is determined by the name of the collector specified in its configuration.

The data written into the diagnostics bundle follows this format:

```
<collector-name> / <node-name> / data / (file1|file2|...)
```

The following is a sample configuration file:

1307 https://github.com/mesosphere/troubleshoot/blob/v0.13-d2iq/README.d2iq.md

```
spec:
  collectors:
    - execCopyFromHost:
        name: node-diagnostics
        image: mesosphere/dkp-diagnostics-node-collector:latest
        timeout: 30s
        command:
          - "/bin/bash"
          - "-c"
          - "/diagnostics/container.sh --hostroot /host --hostpath ${PATH} --
outputroot /output"
        workingDir: "/diagnostics"
        includeControlPlane: true
        privileged: true
        capabilities:
          - AUDIT_CONTROL
          - AUDIT_READ
          - BLOCK_SUSPEND
          - BPF
          - CHECKPOINT_RESTORE
          - DAC_READ_SEARCH
          - IPC_LOCK
          - IPC_OWNER
          - LEASE
          - LINUX_IMMUTABLE
          - MAC_ADMIN
          - MAC_OVERRIDE
          - NET_ADMIN
          - NET_BROADCAST
          - PERFMON
          - SYS_ADMIN
          - SYS_BOOT
          - SYS_MODULE
          - SYS_NICE
          - SYS_PACCT
          - SYS_PTRACE
          - SYS_RAWIO
          - SYS_RESOURCE
          - SYS_TIME
          - SYS_TTY_CONFIG
          - SYSLOG
          - WAKE_ALARM
        extractArchive: true
```

The following is an example of the data produced by running this collector:

```
├── node-diagnostics
│   ├── troubleshoot-control-plane
│   │   └── data
│   │       ├── certs_expiration_kubeadm
```

```
|   |       ├── containerd_config.toml
                ...
|   |       └── whoami_validate
|   └── troubleshoot-worker
|       └── data
|           ├── containerd_config.toml
|           ├── containers_crictl
                ...
|           └── whoami_validate
```

In the event that an error occurs while collecting node diagnostics, the `node-diagnostics/<node>/pod-collector.json` file contains the serialized JSON representations of the running pod. This helps debug the reasons for the collection failure. The `node-diagnostics/<node>/pod-collector.log` file contains stdout from the collector container that runs the diagnostics script. In addition, the command may also produce certain `-error.txt` files. `file-copy-error.txt` and `pod-collector-files-copy-error.txt` are two file examples. These files contain error messages generated while trying to fetch log files from the collector.

When using this collector for node level information you must run additional docker containers and must have the following docker images:

- `mesosphere/pause-alpine:3.2`
- `mesosphere/dkp-diagnostics-node-collector:$(dkp-diagnose version)`

For more information on the configuration options see the `ExecCopyFromHost` in the `pkg/apis/troubleshoot/v1beta2/exec_copy_from_host.go` file.

11.1.3.1.2 `AllLogs` Collector

This collector gathers pod logs from specified namespaces or from all namespaces if none are specified. You can collect logs of all the pods from all the namespaces. The pod logs are collected under the `allPodLogs` directory.

The data written into the diagnostics bundle follows this format:

```
<collector-name> / <namespace-name> / <pod-name> - (container1|container2|...)
```

The following is a sample configuration file to collect logs from all the pods from all the namespaces:

```
spec:
  collectors:
    - allLogs:
        namespaces:
          - "*"
```

The following is a sample configuration file to collect logs from all the pods from specific namespaces:

```
spec:
  collectors:
    - allLogs:
        namespaces:
          - default
          - dev
          - prod
```

The following is an example of the data produced by running this collector:

```
├── node-diagnostics
│   ├── troubleshoot-control-plane
│   │   └── data
│   │       ├── certs_expiration_kubeadm
│   │       ├── containerd_config.toml
│   │           ...
│   │       └── whoami_validate
│   └── troubleshoot-worker
│       └── data
│           ├── containerd_config.toml
│           ├── containers_crictl
│               ...
│           └── whoami_validate
```

In the event that an error occurs while collecting node diagnostics, the `node-diagnostics/<node>/pod-collector.json` file contains the serialized JSON representations of the running pod. This helps debug the reasons for the collection failure. The `node-diagnostics/<node>/pod-collector.log` file contains stdout from the collector container that runs the diagnostics script.

When using this collector for node level information you must run additional docker containers and must have the following docker images:

- `mesosphere/pause-alpine:3.2`
- `mesosphere/dkp-diagnostics-node-collector:$(dkp-diagnose version)`

For more information on the configuration options see the `ExecCopyFromHost` in the `pkg/apis/troubleshoot/v1beta2/exec_copy_from_host.go` file.

11.1.3.1.3 Collect from all Namespaces for `ConfigMap` and `Secret` Collector

Support for collecting from all namespaces for `ConfigMap` and `Secret` collector.

In the original collectors `namespace` there is a required parameter. This adds support for collecting from all namespaces by not setting the `namespace` (or setting it to `""`). Note: To collect all config maps / secrets an empty selector must be used (`selector: [""]`).

11.1.3.1.4 Support for Optional support-bundle Name Prefix

When generating a support bundle, you need naming defaults to provide deterministic bundle identifiers. This feature is especially useful for our convenience extension of providing diagnostics for both, a bootstrap, Konvoy, or other K8s cluster. Using an empty prefix keeps the original naming convention.

11.1.3.1.5 `ClusterResources` Collector

Another customization is added to collect custom resource definitions and all custom resources in the cluster.

11.1.3.2 Next Section

Common Information Discovery Commands (see page 1767)

11.2 Common Information Discovery Commands

When troubleshooting Cluster Operations Management, sometimes just knowing the what, where, status, etc. can help you determine where the problem is and help reveal a solution to an issue.

Several `kubectl` commands can give you information from your environment that can be useful in determining where a problem is occurring.

11.2.1 Application Discovery Commands

Application definitions originate from the source Git repo. These are configured in-cluster for Flux to consume. You can use common `kubectl` verbs to investigate status and gather log data to help determine any problem areas:

- Leverage `kubectl` to investigate the Git repo:
  - ```
    kubectl get gitrepository -A
    ```

    OR
  - ```
    kubectl describe -n <namespace> gitrepository <name>
    ```
- Check status logs of the `source-controller` pod:

- ```
  kubectl describe -n kommander-flux deploy/source-controller
  ```

  OR

  ```
  kubectl logs -n kommander-flux deploy/source-controller --tail -1
  ```

- Verify the status of the associated Apps, ClusterApps and AppDeployments:
  - ```
    kubectl get apps,clusterapps,appdeployments -A
    ```
- Check the status and logs of the `kommander-appmanagement` pod:
  - ```
    kubectl describe -n kommander deploy/kommander-appmanagement
    ```

    OR

    ```
    kubectl logs -n kommander deploy/kommander-appmanagement --tail -1
    ```
- **Kustomizations** -
  - Check the status of all Kustomizations created by AppDeployment:
    - ```
      kubectl get kustomization -A
      ```

      OR

    - ```
      kubectl describe -n <namespace> kustomization <name>
      ```
  - Check into the status and logs of the `kustomize-controller` pod since it manages Kustomizations in the `kommander-flux` Namespace.
    - ```
      kubectl describe -n kommander-flux deploy/kustomize-controller
      ```

      OR

    - ```
      kubectl logs -n kommander-flux deploy/kustomize-controller --tail
      -1
      ```
- **HelmRelease** -
  - HelmRelease(s) are deployed by each Kustomization and for the Application to function, these resources must be in the “Ready” state. To check:
      ```

- `kubectl get helmrelease -A`

  OR

- `kubectl describe -n <namespace> helmrelrease <name>`

- Each HelmRelease also consumes a HelmChart. We can ensure the status of HelmCharts with:
  - `kubectl get helmchart -A`

    OR

  - `kubectl describe -n <namespace> helmchart <name>`
- The controller that manages HelmReleases and HelmCharts is the `helm-controller` in the `kommander-flux` Namespace. It can be helpful to check into the status and logs of the `helm-controller` pod:
  - `kubectl describe -n kommander-flux deploy/helm-controller`

    OR

  - `kubectl logs -n kommander-flux deploy/helm-controller --tail -1`

11.2.1.1 Next Topic

Workspace Discovery Commands (see page 1769)

11.2.2 Workspace Discovery Commands

Workspaces are managed by a Custom Resource named Workspaces. You can use common `kubectl` verbs to investigate status and gather log data to help determine any problem areas:

- Leverage `kubectl` to investigate workspace information:
  - `kubectl get workspaces`

OR

```
kubectl describe workspace <name>
```

- The controller that manages Workspaces is the `kommander-cm` deployment in the `kommander` Namespace. Check status logs :

```
kubectl describe -n kommander deployment/kommander-cm
```

OR

```
kubectl logs -n kommander deployment/kommander-cm --all-containers --
tail -1
```

- **Federated* Resources** -
  - Workspace resources federated to Attached clusters are managed by **Federated* resources**. To view the status of all Federated resources for the Workspace Namespace, we can leverage the following command(s):

```
for resource in $(kubectl api-resources | awk '$1 ~ /
^federated.*$/ {print $1}'); do kubectl get -n <workspace-
namespace> $resource --show-kind --ignore-not-found; done
```

OR

```
kubectl describe -n <workspace-namespace> <resource>
```

  - The controller that manages **Federated* resources** is the `kubefed-controller-manager` in the `kube-federation-system` Namespace. It can be helpful to check into the status and logs of the `kubefed-controller-manager` pod:

```
kubectl describe -n kube-federation-system deployment/kubefed-
controller-manager
```

OR

```
for pod in $(kubectl get po -n kube-federation-system -lkubefed-
control-plane=controller-manager -oname); do kubectl logs -n kube-
federation-system $pod --all-containers --tail -1; done
```

- **Attached Clusters** -
  - To check:

-

  OR

-

- Ensure the status with:

  -

    OR

  -

- It can be helpful to check into the status and logs

  -

    OR

  -

11.2.2.1 Next Section

Application Troubleshooting (see page 1771)

11.3 Application Troubleshooting

Not sure if you are having problems with the Applications deployed on the Kommander component of DKP? This section will explain some behaviors that you may observe if you are having issues with Applications. Also you see context around the machinery that manages the resources and how they interact.

11.3.1 Related Topics

- Deploy Platform Applications via CLI (see page 667)
- Platform Application Dependencies (see page 669)

11.3.2 Applications Context

Application definitions originate from the source Git Repository(ies). They are configured inside the cluster for Flux to consume. The controller that manages GitRepositories is the `source-controller` deployment in the `kommander-flux` **Namespace**.

Kommander Applications are deployed as AppDeployments (see page 644) and reference `Apps` and `ClusterApps`. The controller that manages `Apps`, `ClusterApps`, and `AppDeployments` is the `kommander-appmanagement` deployment in the `kommander` **Namespace**.

Each AppDeployment creates a `Kustomization`. The controller that manages `Kustomizations` is the `kustomize-controller` in the `kommander-flux` **Namespace**.

Each `Kustomization` typically deploys a few resources, with HelmRelease(s) being the most notable. For the application functionality, these resources should always be *Ready*. Each HelmRelease also consumes a HelmChart. The controller that manages HelmReleases and HelmCharts is the `helm-controller` in the `kommander-flux` **Namespace**.

Each DKP release's set of Release Notes will hold the current version information for Applications.

11.3.3 Troubleshooting Approach

In order to properly troubleshoot an issue, you need to be familiar with how all the pieces connected to Applications work. This knowledge will assist in identifying the issue and where to start troubleshooting.

For specific symptoms, you can always search the Support Knowledge Base[1308] for resolutions to behaviors.

Example issues might be:

- *My Kommander Apps are not getting federated down to my attached clusters.*
- *The process has halted, how can I check my status?*

Use the subtopics in this section for more specific help:

- Applications Deployment CLI vs UI (see page 1773)
- Application YAML Customizations (see page 1777)

11.3.3.1 Related Topics

In the Day 2 - Cluster Operations Management (see page 590) section of the Documentation, there are entire sections on Platform Applications (see page 662), Workspaces (see page 678) and more that will give further context for all these quicker troubleshooting customization tips.

1308 https://d2iqhelp.zendesk.com/agent/dashboard

11.3.3.2 Next Topic

Applications Deployment CLI vs UI (see page 1773)

11.3.4 Applications Deployment CLI vs UI

When installing DKP, an `AppDeployment` resource is created for each enabled **Platform Application**. This `AppDeployment` resource references a `ClusterApp`, which then references the repository that contains a concrete declarative and preconfigured setup of an application, usually in the form of a `HelmRelease`. `ClusterApps` are cluster-scoped so that these Platform Applications are deployable to all Workspaces or Projects. Also refer to AppDeployment Resources (see page 644) from the Day 2 section of the Documentation.

Applications can be enabled and configured using the UI or CLI at both levels:

- **Workspace level** for all clusters in a Workspace
- **Cluster level** for specific clusters *within* a Workspace.

See the following table for an overview:

Desired level of configuration	**CLI**	**UI**
Cluster-scoped deployment	Create an AppDeployment, and specify target clusters in `spec.clusterSelector`. Example deployment (see page 684)	Go to target workspace and enable through the application card, selecting the target clusters. Example deployment (see page 660)
Workspace-scoped deployment	Create an AppDeployment without a `spec.clusterSelector` section. Automatically deploys to all clusters in the workspace. Example deployment (see page 0)	Go to target workspace and enable through the application card, selecting all target clusters. Example deployment (see page 661)
Cluster-scoped customization	Create an AppDeployment, and specify target clusters in `spec.clusterSelector`. Reference `ConfigMap` with customization overrides in `clusterConfigOverrides`. Example deployment (see page 685)	Go to target workspace and enable through the application card. Select the target clusters, and establish customizations in the configuration service per cluster. Example deployment (see page 661)

Desired level of configuration	CLI	UI
Workspace-scoped customization	Create an AppDeployment without a `spec.clusterSelector` section. Reference `ConfigMap` with customization overrides in `clusterConfigOverrides`. Automatically deploys customization to all clusters in the workspace. Example deployment (see page 644)	Go to target workspace and enable through the application card. Manually select all clusters and copy-paste the customization in the configuration service for all clusters. Example deployment (see page 661)

11.3.4.1 Next Topic

Applications Deployment in a Workspace (see page 1774)

11.3.4.2 Applications Deployment in a Workspace

11.3.4.2.1 Print and Review the Current State of an `AppDeployment` Resource

If you want to know how the `AppDeployment` resource is currently configured, use the commands below to print a table of the declared information. If the `AppDeployment` is configured for several clusters in a workspace, a column will display a list of the clusters.

11.3.4.2.1.1 Review all `AppDeployments` in a workspace

To review the state of the `AppDeployment` resource for a specific workspace, run the `get` command with the name of your workspace, as in this example:

```
dkp get appdeployments -w kommander-workspace
```

The output should contain a list of all your applications, here is an example:

```
NAME APP CLUSTERS
[...]
kube-oidc-proxy                kube-oidc-proxy-0.3.2           host-cluster
kube-prometheus-stack          kube-prometheus-stack-46.8.0    host-cluster
```

```
kubecost                        kubecost-0.35.1                 host-cluster
[...]
```

11.3.4.2.1.2 Review a specific `AppDeployment` of an application in a workspace

To review the state of a specific `AppDeployment` of an application, run the `get` command with the name of the application and your workspace, as in this example:

```
dkp get appdeployment kube-prometheus-stack -w kommander-workspace
```

The output should look similar to this:

```
NAME APP CLUSTERS
kube-prometheus-stack kube-prometheus-stack-46.8.0 host-cluster
```

11.3.4.2.2 Deployment Scope

In a single-cluster environment with an **Essential license**, `AppDeployments` enable customizing any platform application.

In a multi-cluster environment with an **Enterprise license**, `AppDeployments` enable workspace-level (see page 124), project-level (see page 720), and per-cluster deployment and customization of workspace applications (see page 682).

11.3.4.2.3 Use Case Example

As a user, I have multiple clusters in a *Workspace*. I want to enable an application into some, but not all clusters in that workspace. I also want to enable a custom configuration.

To define which cluster should have an Application deployed onto them, use the AppDeployment's `spec.clusterSelector` field.

The user can edit the AppDeployment at any time to add or remove clusters from the spec.

Enable *Workspace* Applications for a subset of clusters and custom configurations for each cluster:

1. Create an AppDeployment using `dkp create appdeployment` to enable and disable clusters, and set cluster config overrides per cluster:

   ```
   dkp create appdeployment
   ```

2. Use the code editor to directly edit the AppDeployment spec to:
    a. Enable or disable clusters
    b. Add or remove cluster config overrides for clusters

    ```
    dkp edit appdeployment APPDEPLOYMENT_NAME -n NAMESPACE
    ```

    EX: `dkp edit appdeployment kube-oidc-proxy -n kommander`

> `NAMESPACE` is the namespace in which the AppDeployment resides - this can be a Workspace or a Project Namespace. To list all available workspace namespaces, run `kubectl get workspaces`.

3. **TROUBLESHOOT**: For up-to-date status on which clusters an application has been enabled, or what configuration overrides have been applied to each cluster, check the AppDeployment status:

    ```
    dkp get appdeployment
    ```

4. Use to get the entire YAML object, including cluster config overrides that might be set on a cluster - similar to `kubectl get appdeployment`.

    ```
    dkp get appdeployment --output yaml
    ```

    Example Output:

    ```
    apiversion: apps.kommander.d2iq.io/v1alpha3
    kind: AppDeployment
    metadata:
      generation: 1  # this means there is only one instance of the application,
    must match with the number of instances you expect to see
      name: kube-oidc-proxy  # APP ID of the application as displayed in the
    Release Notes
      namespace: kommander # equal to the workspace namespace of the workspace
    where the app is deployed
    spec:
      appRef:
        kind: ClusterApp
        name: kube-oidc-proxy-0.3.2 # APP ID with the app version as displayed in
    the Release Notes
      clusterConfigOverrides:  # present only for apps with custom configurations
      - clusterSelector:   # present only for workspaces with several clusters
          matchLabels:
    ```

```
        kommander.d2iq.io/cluster-name: host-cluster  # cluster name where app
is deployed
    configMapName: kube-oidc-proxy-host-cluster  # name of the ConfigMap file
that contains the Overrides with the custom configuration for this specific
cluster
  clusterSelector:
    matchExpressions:
    - key: kommander.d2iq.io/cluster-name
      operator: In
      values:
      - host-cluster # cluster name where app is deployed
status:
  clusters:
  - clusterConfigOverridesRef:
    name: kube-oidc-proxy-host-cluster # name of the ConfigMap file that
contains the Overrides with the custom configuration for this specific cluster
  conditions:
  - status: "True" # must be True for app to be deployed successfully
    type: AppDeploymentEnabled
  name: host-cluster
observedGeneration: 1
```

11.3.4.2.3.1 Verify Applications

The applications are now enabled. Connect to the attached cluster and check the `HelmReleases` to verify the deployment:

```
kubectl get helmreleases -n ${WORKSPACE_NAMESPACE}
```

The output looks similar to this:

```
NAMESPACE              NAME              READY   STATUS
AGE
workspace-test-vjsfq   kafka-operator    True    Release reconciliation succeeded
7m3s
```

11.3.4.2.3.2 Next Topic

Application YAML Customizations (see page 1777)

11.4 Application YAML Customizations

DKP provides the ability to configure an application for all clusters in a workspace, but also for a subset of clusters in a workspace. You can also apply customizations per cluster or workspace.

An `AppDeployment` object creates a local HelmRelease YAML file that establishes how an Application is deployed. To further customiza the application, you can then create a `ConfigMap` object with overriding configuration.

You can apply one and the same `ConfigMap` to several clusters in a workspace, or choose to create different customizations in individual `ConfigMaps` for each cluster.

The Appdeployment specifies the name of the app, the version of the app, the workspace where it applies Additionally, you can specify a subset of clusters, where it should be deployed, and if required, any customizations.

For example, this is the **default** `AppDeployment` for the Kube Prometheus Stack Platform Application:

```
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: kube-prometheus-stack-46.8.0
    kind: ClusterApp
```

11.4.1 Override ALL Clusters within a Workspace

If you want to change the default configuration (customize an application), add a `ConfigMap` with your override values. Then add that reference to the `AppDeployment` by setting the `configOverrides` field to the name of that `ConfigMap`. When it refreshes the HelmRelease, it will add references to those ConfigMaps. This overrides the configuration of the app for all clusters *within* the Workspace, unless specified differently in the AppDpeloyment though the `spec.clusterSelector`.

Example #1:

```
kind: HelmRelease
metadata:
  ...
  name: dex
  namespace: kommander
...
spec:
...
  valuesFrom:
  - kind: ConfigMap
    name: dex-2.13.5-d2iq-defaults
  - kind: ConfigMap
    name: dex-cluster-overrides
```

Example #2:

This is an example, of how to customize the `AppDeployment` of Kube Prometheus Stack:

1. Provide the name of a `ConfigMap` with the custom configuration in the `AppDeployment`:

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: kube-prometheus-stack-46.8.0
    kind: ClusterApp
  configOverrides:
    name: kube-prometheus-stack-overrides-attached
EOF
```

   Line 12 references the name of the ConfigMap with the customization.

2. Create the `ConfigMap` with the name from the previous step, which provides the *custom* configuration on top of the *default* configuration:

```
cat <<EOF > kube-prometheus-stack-overrides-attached | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: kube-prometheus-stack-overrides-attached
data:
  values.yaml: |
    prometheus:
      prometheusSpec:
        storageSpec:
          volumeClaimTemplate:
            spec:
              resources:
                requests:
                  storage: 150Gi
EOF
```

11.4.2 Subtopics

11.4.3 Available Customizations

Application YAML can be customized, but which customizations can you do and why would you customize? Take a look below for the answers.

11.4.3.1 Why Customize?

Your environment and provider combined with various image locations or security requirements might dictate the why behind customizing your Applications.

- Enable certain images
- Provide more storage
- Bump memory resources

Depending on which level you need your customizations, determines the “how”. Refer to the pages that follow for specifics about how to customize your Applications.

11.4.3.2 How do I Customize?

Number one, create a `ConfigMap`. A `ConfigMap` allows you to decouple environment-specific configuration from your container images[1309], so that your applications are easily portable. Use a `ConfigMap` to set configuration data that is separate from the Application code. Provide the name of a `ConfigMap` in the `AppDeployment`, which provides custom configuration on top of the default configuration.

This is an example, of how to customize the `AppDeployment` of Kube Prometheus Stack:

1. Create the `ConfigMap` which provides the custom configuration on top of the default configuration:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: ${WORKSPACE_NAMESPACE}
  name: kube-prometheus-stack-overrides-attached
data:
  values.yaml: |
    prometheus:
      prometheusSpec:
        storageSpec:
          volumeClaimTemplate:
            spec:
              resources:
                requests:
```

1309 https://kubernetes.io/docs/reference/glossary/?all=true#term-image

```
                    storage: 150Gi
EOF
```

2. Provide the name of a `ConfigMap` with the custom configuration in the `AppDeployment`. Override the default configuration of an Application by setting the `configOverrides` field on the `AppDeployment` to that `ConfigMap`.

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: kube-prometheus-stack-46.8.0
    kind: ClusterApp
  configOverrides:
    name: kube-prometheus-stack-overrides-attached
EOF
```

- The changes are applied only if the YAML file has a valid syntax.
- Set up only one cluster override `ConfigMap` per cluster. If there are several `ConfigMaps` configured for a cluster, only one will be applied.
- Cluster override `ConfigMaps` must be created on the Management cluster.

11.4.3.2.1 Related Topics

- Customize an Application per Cluster (see page 685)
- Disable a Custom Configuration of an Application for a Cluster (see page 688)

11.4.3.2.2 Next Topic

Enable or Disable an App per Cluster (see page 1781)

11.4.4 Enable or Disable an App per Cluster

For clusters that are newly attached into the workspace, all Applications enabled for the Workspace are automatically enabled on and deployed to the new clusters. If you want to see on what clusters your application is currently deployed, follow the steps below.

11.4.4.1 Print and Review the Current State of an `AppDeployment` Resource

If you want to know how the `AppDeployment` resource is currently configured, use the commands below to print a table of the declared information. If the `AppDeployment` is configured for several clusters in a workspace, a column will display a list of the clusters.

11.4.4.1.1 Review all `AppDeployments` in a workspace

To review the state of the `AppDeployment` resource for a specific workspace, run the `get` command with the name of your workspace, as in this example:

```
dkp get appdeployments -w kommander-workspace
```

The output should contain a list of all your applications, here is an example:

```
NAME APP CLUSTERS
[...]
kube-oidc-proxy                 kube-oidc-proxy-0.3.2           host-cluster
kube-prometheus-stack           kube-prometheus-stack-46.8.0    host-cluster
kubecost                        kubecost-0.35.1                 host-cluster
[...]
```

11.4.4.1.2 Review a specific `AppDeployment` of an application in a workspace

To review the state of a specific `AppDeployment` of an application, run the `get` command with the name of the application and your workspace, as in this example:

```
dkp get appdeployment kube-prometheus-stack -w kommander-workspace
```

The output should look similar to this:

```
NAME APP CLUSTERS
kube-prometheus-stack kube-prometheus-stack-46.8.0 host-cluster
```

11.4.4.2 Deployment Scope

In a single-cluster environment with an **Essential license**, `AppDeployments` enable customizing any platform application.

In a multi-cluster environment with an **Enterprise license**, `AppDeployments` enable workspace-level (see page 124), project-level (see page 720), and per-cluster deployment and customization of workspace applications (see page 682).

11.4.4.3 Define a Custom Configuration

Enable or Disable an Application per Cluster ***after it has been enabled at the workspace level***. You can enable or disable applications at any time. After you have enabled the application at the workspace level (see page 0), the `spec.clusterSelector` field (see page 1795) populates.

Edit the `AppDeployment` YAML by adding or removing the names of the clusters where you want to enable your application in the `clusterSelector` section:

The following snippet is an example. To customize, replace the following according to your requirements:

- Application name
- Version
- Workspace name
- Cluster names

See Components and Applications[1310] for the compatible versions.

```
cat <<EOF | kubectl apply -f -
apiVersion: apps.kommander.d2iq.io/v1alpha3
kind: AppDeployment
metadata:
  name: kube-prometheus-stack
  namespace: ${WORKSPACE_NAMESPACE}
spec:
  appRef:
    name: kube-prometheus-stack-46.8.0
    kind: ClusterApp
  clusterSelector:
    matchExpressions:
    - key: kommander.d2iq.io/cluster-name
      operator: In
      values:
      - attached-cluster1
      - attached-cluster3-new
EOF
```

11.4.4.3.1 Verify the Current Configuration of your Application

Connect to the Managed or Attached cluster and check the `HelmReleases` and status to verify the deployment:

1310 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/214630420/DKP+2.6.0+Components+and+Applications#applications

```
kubectl get helmreleases -n ${WORKSPACE_NAMESPACE}
```

The output looks similar to this:

```
NAMESPACE              NAME              READY   STATUS
AGE
workspace-test-vjsfq   kafka-operator    True    Release reconciliation succeeded
7m3s
```

11.4.4.3.2 Next Topic

Workspace Application per Cluster Enablement (see page 1784)

11.4.5 Workspace Application per Cluster Enablement

DKP provides the ability to configure an application for all clusters in a workspace, but also for a subset of clusters in a workspace. You can also apply customizations per cluster or workspace.

See Applications Deployment (see page 1773) for an overview of the configuration options.

- The layers of configuration consist of:
  - Default application config
  - Workspace config overrides
  - Cluster config overrides.

Each layer of configuration applies on top of the previous one.

11.4.5.1 Example of a Per Cluster Deployment:

In this example AppDeployment YAML, the fields are defined below the output:

```
dkp create appdeployment reloader
    --workspace default-workspace
    --app reloader-0.0.110
    --clusters attached-cluster1,attached-cluster2
```

- `reloader` - reloader[1311] is a controller that watches changes on ConfigMaps and Secrets, and automatically triggers updates on the dependent applications.

1311 https://github.com/stakater/Reloader

- if `--clusters` is omitted, all clusters in the Workspace are enabled. The resulting `AppDeployment` spec would include all of the Workspace's clusters' names.
- `attached-cluster1` and `attached-cluster2` are `KommanderCluster` names. (The `kommander.mesosphere.io/cluster-name` label is set on all `KommanderCluster` objects)

Output:

```
apiVersion: apps.kommander.d2iq.io/v1alpha2
kind: AppDeployment
metadata:
  name: reloader
  namespace: kommander-default-workspace
spec:
  appRef:
    kind: ClusterApp
    name: reloader-0.0.110
  configOverrides:
    name: reloader-ws-overrides
  clusterSelector:
    matchExpressions:
      - {key: kommander.mesosphere.io/cluster-name, operator: In, values: [attached-
cluster1, attached-cluster2]}
```

11.4.5.2 Field Definitions:

- `Clusters` - tracks clusters that have the AppDeployment enabled along with a ref to the cluster config overrides
  - configmap in the management cluster
  - Existence of `cluster` means that the cluster's default AppDeployment state has been processed.
  - Nonexistence of `cluster` means that the cluster was newly attached and needs to be selected by default to enable the AppDeployment on it.
- `AppRef` - provides reference to a ClusterApp (platform applications) or App (catalog or custom applications) object.
- `ConfigOverrides` - allows a user to reference a ConfigMap that contains configuration overrides
- `ClusterSelector` - defines the label selectors to select clusters on which to enable the AppDeployment
- `ClusterConfigOverrides` - defines a list of config overrides ConfigMaps and label selectors to select clusters on which to apply the config overrides
- `ClusterConfigOverridesRef` - is set to reference the overrides ConfigMap in the Management cluster

NOT SURE WHERE THIS EXAMPLE CAME FROM

```
status:
  clusters:
    - name: attached-cluster1
      clusterConfigOverridesRef: reloader-cluster1-overrides
      conditions:
      - lastTransitionTime: "2022-04-21T23:07:47Z"
        status: "True"
        type: AppDeploymentEnabled
    - name: attached-cluster2
      clusterConfigOverridesRef: reloader-cluster2-overrides
      conditions:
      - lastTransitionTime: "2022-04-21T23:07:47Z"
        status: "True"
        type: AppDeploymentEnabled
    - name: attached-cluster3
      conditions:
      - lastTransitionTime: "2022-04-21T23:07:47Z"
        status: "False"
        type: AppDeploymentEnabled
```

11.5 Object Locations and Explanations

Objects are created and interact in many steps with coordination between various components and applications. What deploys the Platform Applications, where are the charts stored, what is the mechanism that deploys them and how are settings customized.

After using the DKP CLI to install the Konvoy (see page 78) components of DKP, install the Kommander component (see page 78).

11.5.1 Kommander

Kommander uses many individual Applications for management of resources. Below is an explanation of the process(es) that deploys them.

11.5.1.1 Objects and the Resources they Manage

- `kommander-applications` repository[1312] manages:
  - **Flux** - Deploys Flux from a set of Kubernetes manifest files and
  - **Helm Repositories** applies the Helm Repository resources from this repository.
  - **Gitea** - Creates a secret that hold the Gitea user credentials as well as deploys Gitea by applying the HelmRelease directly to the cluster. Furthermore, it creates a Gitea admin user and the Kommander Git repository.

1312 https://github.com/mesosphere/kommander-applications/tree/main/common/helm-repositories

    - **Applications Definitions** then populate the management Git repository hosted by Gitea by copying the local `kommander-applications` repo provided via the `--kommander-applications-repository` flag.
    - **ChartMuseum** - *Air-gapped Only*, ChartMuseum is used on air-gapped installations to store the Helm Charts for Air-gapped installations. In non-air-gapped installations, the charts are fetched from upstream repositories and ChartMuseum is not installed.
    - **Gatekeeper** - Deploys Gatekeeper by applying the HelmRelease directly to the cluster. The HelmRelease is adopted by AppDeployment.
- `kommander-appmanagement` manages:
    - **AppManagement** - Deploys `kommander-appmanagement` via applying the HelmRelease directly to the cluster.
    - **Core AppDeployments** - Creates AppDeployments for the core components: Flux, `kommander-appmanagement` and ChartMuseum. These components were previously deployed, but this creates the AppDeployments which will eventually manage the Applications.
- `apps-kommander` manages:
    - **Bootstrap Repository -** creates the `apps-kommander` Kubstomization which deploys apps into the `kommander` Namespace on the Management Cluster. Commits manifests and directories to the Management git repository to setup the Management Cluster and the repository structure.
- `kommander-ca` manages:
    - **Root CA** which creates Cert-manager custom resources: `kommander-ca` clusterIssuer, `kommander-ca` Certificate, and a self-signed ClusterIssuer. Then waits for the `kommander-ca` Certificate to be created and accessible.
- `kommander` manages:
    - **Ingress Certificate** which uses chart values to create the self-attached KommanderCluster with custom domain/custom certificate settings. Also creates ACME resources(ClusterIssuer) as defined in the installer configuration.
- `kommander-vars` manages:
    - **Flux Configuration** which creates the management GitRepository object and commits it to Git as well as encrypts and commits the Gitea credentials to Git. Flux populates the `kommander-vars` ConfigMap with substitution variables.
    - **Catalog Repository Loader** - Creates and commits the GitRepository objects to the management Git repo that are defined in the installer config(catalog). These default catalo Git repositories are then managed by a controller (DefaultCatalogGitRepository) which does things like:
        i. Propagate the GitRepository objects to workspaces and projects based on labels
        ii. Handles updating the GitRepository URL to use the correect hostname so the attached clusters can communicate with Gitea.

11.5.1.2 Next Topic

Kommander Resource Locations (see page 1788)

11.5.2 Kommander Resource Locations

Installation happens in many steps with coordination between various components and applications. This document explains the process and locations for various repositories. After using the DKP CLI to install the Konvoy components, install the Kommander component (see page 78) and the following process begins.

11.5.2.1 Kommander Installation Part I:

1. **Base** - Deploys base resources required including Kommander and `kommander flux` namespaces.
2. **Flux** - Deploys Flux from a set of Kubernetes manifest files in the `kommander-applications` repository (deployments, etc).
3. **Root CA** - Creates Cert-manager custom resources: `kommander-ca` clusterIssuer, `kommander-ca` Certificate, and a self-signed ClusterIssuer. Then waits for the `kommander-ca` Certificate to be created and accessible.
4. **ChartMuseum** - Installs via the Helm library. The ChartMuseum Helm Release is created but will reconcile when the Git repository is created and populated. This is due to ChartMuseum being stored in the GitRepository. The installer will also set the helmRepositoryURL in the internal installer configuration to be used during Helm repository installation next.
5. **Helm Repositories** - Applies the Helm Repository resources from the `kommander-applications` repo https://github.com/mesosphere/kommander-applications/tree/main/common/help-repositories[1313]. If the helmRepositoryURL was set in previous step, it will be applied via a substitution variable when teh Helm Repository objects are applied. EX: "${helmMirrorURL:=https://kubernetes-charts.banzaicloud.com/}"
6. **Ingress Certificate** - Sets `kommander` chart values to create the self-attached KommanderCluster with custom domain/custom certificate settings. Also creates ACME resources(ClusterIssuer) as defined in the installer configuration.
7. **Gitea** - Creates a secret that hold the Gitea user credentials as well as deploys Gitea by applying the HelmRelease directly to the cluster. Furthermore, it creates a Gitea admin user and the Kommander git repository.
8. **Applications Definitions** - Populates the management Git repository hosted by Gitea by copying the local kommander-applications repo provided viat the `--kommander-applications-repository` flag.

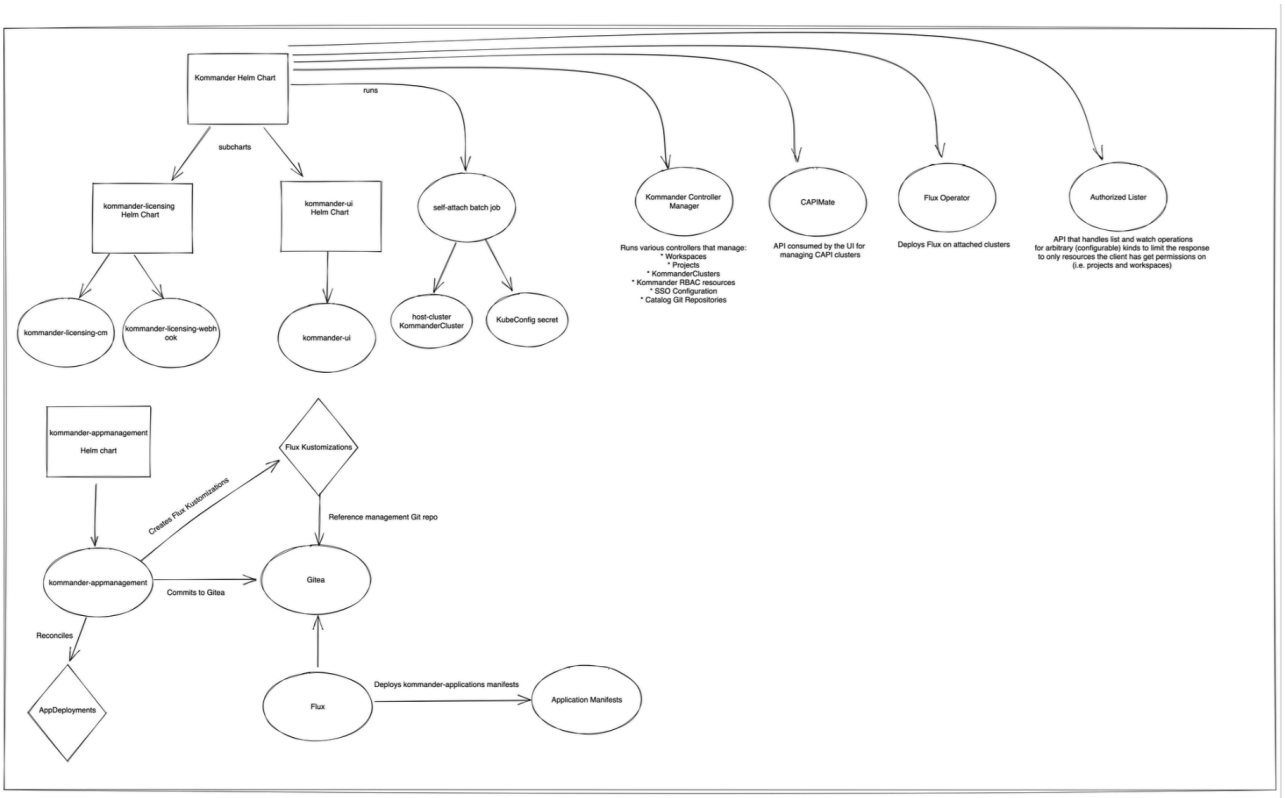
1313 https://github.com/mesosphere/kommander-applications/tree/main/common/helm-repositories

9. **Bootstrap Repository** - Commits manifests and directories to the Management git repository to setup the Management Cluster and the repository structure. It also creates the `apps-kommander` Kubstomization which deploys apps into the `kommander` namespace on the Mangement Cluster.
10. **Age** - Create Age Encryption section for SOPs encryption and adds the Age recipient(public) key to git.
11. **Flux Configuration** - Populates the `kommander-vars` ConfigMap with substitution variables as well as creates the management GitRepository object and commits it to Git. Also encrypts and commits the Gitea credentials to Git.
12. **Gatekeeper** - Deploys Gatekeeper by applying the HelmRelease directly to the cluster. The HelmRelease is adopted by AppDeployment.
13. **AppManagement** - Deploys `kommander-appmanagement` via applying the HelmRelease directly to the cluster.
14. **Core AppDeployments** - Creates AppDeployments for the core components: Flux, `kommander-appmanagement` and ChartMuseum. These components were previously deployed, but this creates the AppDeployments which will eventually manage the Applications.
15. **Optional AppDeployments** - Creates all the AppDeployments for the applications defined in the installer configuration.
16. **Catalog Repository Loader** - Creates and commits the GitRepository objects to the management Git repo that are defined in the installer config(catalog). These default catalo Git repositories are then managed by a controller (DefaultCatalogGitRepository) which does things like:
    a. Propagate the GitRepository objects to workspaces and projects based on labels
    b. Handles updating the GitRepository URL to use the correect hostname so the attached clusters can communicate with Gitea.

Each DKP release's set of Release Notes will hold the current version information for Applications.

11.5.2.2 Kommander Installation Part II:

This part of the installation deals with Kommander Controller and Helm Charts.



11.5.2.3 Next Section

Konvoy Image Builder(KIB) Troubleshooting[1314]

1314 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/315129909

12 CLI and API Tools

CLI commands and API tools.

12.1 API Documentation

This document is automatically generated from the API definition in the code.

12.1.1 apps.kommander.mesosphere.io/v1alpha2

12.1.1.1 App

App is the Schema for a Service or Application in Kommander.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1315]	false
spec		GenericAppSpec (see page 1794)	false
status		object	false

12.1.1.2 AppDeployment

AppDeployment is the Schema for a concrete installation of a Service or Application in Kommander.

1315 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1316]	false
spec		AppDeploymentSpec (see page 1792)	false
status		object	false

12.1.1.3 AppDeploymentList

AppDeploymentList contains a list of AppDeployments.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1317]	false
items		[...]AppDeployment (see page 1791)	true

12.1.1.4 AppDeploymentSpec

AppDeploymentSpec defines an instance of an Application.

Field	Description	Scheme	Required
appRef	AppRef provides reference to a ClusterApp or App object.	TypedLocalObjectReference (see page 1795)	true
configOverrides	ConfigOverrides allows a user to define a ConfigMap that contains configuration overrides	corev1.LocalObjectReference[1318]	false

1316 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1317 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1318 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

12.1.1.5 AppList

AppList contains a list of Apps.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1319]	false
items		[...]App (see page 1791)	true

12.1.1.6 ClusterApp

ClusterApp is the Schema for a Service or Application in Kommander.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1320]	false
spec		GenericAppSpec (see page 1794)	false
status		object	false

12.1.1.7 ClusterAppList

ClusterAppList contains a list of ClusterApps.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1321]	false
items		[...]ClusterApp (see page 1793)	true

1319 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1320 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1321 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

12.1.1.8 CrossNamespaceGitRepositoryReference

CrossNamespaceGitRepositoryReference contains enough information to let you locate the typed referenced object at cluster level.

Field	Description	Scheme	Required
apiVersion	API version of the referent	string	false
kind	Kind of the referent	string	true
name	Name of the referent	string	true
namespace	Namespace of the referent, defaults to the Kustomization namespace	string	false

12.1.1.9 GenericAppSpec

GenericAppSpec defines the actual Application spec.

Field	Description	Scheme	Required
appId	AppID specifies the name of the application/workload	string	true
gitRepositoryRef	GitRepository is reference to the Flux's GitRepository, which in turn describes Git repository where the service resides	CrossNamespaceGitRepositoryReference (see page 1794)	true
version	Version depicts the version of the service in the semantic versioning format.	string	true

12.1.1.10 TypedLocalObjectReference

TypedLocalObjectReference contains enough information to let you locate the typed referenced object at cluster or local level.

Field	Description	Scheme	Required
apiVersion	API version of the referent	string	false
kind	Kind of the referent	string	true
name	Name of the referent	string	true

12.1.2 apps.kommander.mesosphere.io/v1alpha3

12.1.2.1 App

App is the Schema for a Service or Application in Kommander.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1322]	false
spec		GenericAppSpec (see page 1801)	false
status		object	false

12.1.2.2 AppDeployment

AppDeployment is the Schema for a concrete installation of a Service or Application in Kommander.

1322 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1323]	false
spec		AppDeploymentSpec (see page 1797)	false
status		object	false

12.1.2.3 AppDeploymentClusterCondition

Field	Description	Scheme	Required
lastTransitionTime	LastTransitionTime is the last time the condition transitioned from one status to another.	metav1.Time[1324]	false
status	Status of the condition, one of True, False, Unknown	string	true
type	Type indicates type of condition in CamelCase or in foo.example.com/CamelCase.	string	true

12.1.2.4 AppDeploymentList

AppDeploymentList contains a list of AppDeployments.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1325]	false

1323 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1324 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#time-v1-meta
1325 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

Field	Description	Scheme	Required
items		[...]AppDeployment (see page 1795)	true

12.1.2.5 AppDeploymentSpec

AppDeploymentSpec defines an instance of an Application.

Field	Description	Scheme	Required
appRef	AppRef provides reference to a ClusterApp or App object.	TypedLocalObjectReference (see page 1801)	true
clusterConfigOverrides	ClusterConfigOverrides defines a list of config overrides configmaps and label selectors to select clusters on which to apply the config overrides.	[...]ClusterConfigOverrides (see page 1799)	false
clusterSelector	ClusterSelector defines the label selectors to select clusters on which to enable the AppDeployment.	metav1.LabelSelector[1326]	false
configOverrides	ConfigOverrides allows a user to define a ConfigMap that contains configuration overrides at the workspace level.	corev1.LocalObjectReference[1327]	false

12.1.2.6 AppDeploymentStatus

AppDeploymentStatus defines the current state of the AppDeployment.

1326 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#labelselector-v1-meta
1327 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

Field	Description	Scheme	Required
clusters	Clusters tracks clusters that have the AppDeployment enabled along with a ref to the cluster config overrides configmap in the management cluster. Existence of cluster means that the cluster's default AppDeployment state has been processed. Nonexistence of cluster means that the cluster was newly attached and needs to be selected by default to enable the AppDeployment on it.	[...]ClusterDeploymentStatus (see page 1800)	false
observedGeneration	ObservedGeneration is the most recent generation observed for this AppDeployment. It corresponds to the AppDeployment's generation, which is updated on mutation by the API Server.	int64	false

12.1.2.7 AppList

AppList contains a list of Apps.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1328]	false
items		[...]App (see page 1795)	true

1328 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

12.1.2.8 ClusterApp

ClusterApp is the Schema for a Service or Application in Kommander.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1329]	false
spec		GenericAppSpec (see page 1801)	false
status		object	false

12.1.2.9 ClusterAppList

ClusterAppList contains a list of ClusterApps.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1330]	false
items		[...]ClusterApp (see page 1799)	true

12.1.2.10 ClusterConfigOverrides

ClusterConfigOverrides defines the cluster config overrides configmap and label selector to select clusters on which to apply the overrides configmap.

Field	Description	Scheme	Required
clusterSelector	ClusterSelector defines the label selectors to select clusters on which to deploy the configmap.	metav1.LabelSelector[1331]	false

1329 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1330 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1331 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#labelselector-v1-meta

Field	Description	Scheme	Required
configMapName	ConfigMapName is the name of the cluster config overrides configmap.	string	true

12.1.2.11 ClusterDeploymentStatus

Field	Description	Scheme	Required
clusterConfigOverrides Ref	ClusterConfigOverrides Ref is set to reference the overrides ConfigMap in the management cluster	corev1.LocalObjectReference[1332]	false
conditions		[...]AppDeploymentClusterCondition (see page 1796)	false
name		string	true

12.1.2.12 CrossNamespaceGitRepositoryReference

CrossNamespaceGitRepositoryReference contains enough information to let you locate the typed referenced object at cluster level.

Field	Description	Scheme	Required
apiVersion	API version of the referent	string	false
kind	Kind of the referent	string	true
name	Name of the referent	string	true

1332 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

Field	Description	Scheme	Required
namespace	Namespace of the referent, defaults to the Kustomization namespace	string	false

12.1.2.13 GenericAppSpec

GenericAppSpec defines the actual Application spec.

Field	Description	Scheme	Required
appId	AppID specifies the name of the application/workload	string	true
gitRepositoryRef	GitRepository is reference to the Flux's GitRepository, which in turn describes Git repository where the service resides	CrossNamespaceGitRepositoryReference (see page 1800)	true
version	Version depicts the version of the service in the semantic versioning format.	string	true

12.1.2.14 TypedLocalObjectReference

TypedLocalObjectReference contains enough information to let you locate the typed referenced object at cluster or local level.

Field	Description	Scheme	Required
apiVersion	API version of the referent	string	false
kind	Kind of the referent	string	true

Field	Description	Scheme	Required
name	Name of the referent	string	true

12.1.3 dispatch.d2iq.io/v1alpha2

12.1.3.1 GitopsRepository

GitopsRepository represents an SCM repository that backs a gitops resource. A gitops resource can be a FluxCD HelmRelease or Kustomization resource.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1333]	false
spec		GitopsRepositorySpec (see page 1802)	true

12.1.3.2 GitopsRepositoryList

GitopsRepositoryList contains a list of Repository.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1334]	false
items		[...]GitopsRepository (see page 1802)	true

12.1.3.3 GitopsRepositorySpec

GitopsRepositorySpec defines the desired state of GitopsRepository.

1333 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1334 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

Field	Description	Scheme	Required
cloneUrl	URL to clone the repository from.	string	false
secret	Reference to the secret to use when interacting with the Git provider API. The secret should contain the following fields: username: the username (if password is not a token). password: the password or token (required).	string	false
template	Template of the gitops resource backing the repository.	GitopsRolloutTemplate (see page 1803)	false

12.1.3.4 GitopsRolloutTemplate

Field	Description	Scheme	Required
path	Root path to fetch manifests from in the Git repository.	string	false
ref	The Git reference to checkout and monitor for changes, defaults to master branch. This field supersedes the `revision` field in v1alpha1 API and is a breaking change.	sourcectrlv1.GitRepositoryRef	false
suspend	Whether to suspend periodic or webhook-notified sync.	bool	false

12.1.4 kommander.mesosphere.io/v1beta1

12.1.4.1 CAPIClusterReference

Field	Description	Scheme	Required
name		string	true
namespace		string	false

12.1.4.2 ClusterReference

ClusterReference holds a single reference to clusters provisioned via Kommander. Only one field is allowed to be set. Currently, only CAPI clusters are creatable, but this is left extensible for other provider types in the future.

Field	Description	Scheme	Required
capiCluster		CAPIClusterReference (see page 1804)	false

12.1.4.3 GenericClusterReference

GenericClusterReference sets the name of the cluster.

Field	Description	Scheme	Required
name		string	true

12.1.4.4 IngressSpec

IngressSpec holds settings for the cluster's Kommander managed Ingress.

Field	Description	Scheme	Required
certificateSecretRef	CertificateSecretRef is a reference to a secret of type TLS that holds a TLS certificate, private key and CA certificate. The certificate must be valid for the Ingress hostname. The secret must be located in the workspace's namespace on the target cluster.	corev1.LocalObjectReference[1335]	false
hostname	Hostname can be set to configure the cluster's Ingress with a custom domain name.	string	false
issuerRef	IssuerRef is a reference to an `Issuer` or `ClusterIssuer` on the target cluster to be used to create a `Certificate` for the cluster's Ingress. If the type is an Issuer, it must be located in the workspace's namespace on the cluster.	IssuerReference (see page 1806)	false

12.1.4.5 IngressStatus

IngressSpec holds information about the cluster's Kommander managed Ingress.

1335 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

Field	Description	Scheme	Required
address	Address is the main DNS name or IP address of the cluster's Ingress that should be used to access services on the cluster. If the field is empty, it means that the load balancer hasn't been populated yet.	string	false
caBundle	CABundle is a PEM encoded CA bundle which should be used to verify the TLS connection for the Ingress-served end-entity certificate.	[...]byte	false
useSystemCA	UseSystemCA is set to true if the Ingress end-entity certificate was issued by a public CA and it is expected that the root CA cert is included in a OS CA bundle (which should be used for TLS verification in that case). If this field is `true` the `CABundle` field is empty. default: false	bool	false

12.1.4.6 IssuerReference

IssuerReference is a reference to an issuer with a given name, kind and group.

Field	Description	Scheme	Required
group	Group of the issuer being referred to.	string	false

Field	Description	Scheme	Required
kind	Kind of the issuer being referred to.	string	false
name	Name of the issuer being referred to.	string	true

12.1.4.7 KommanderCluster

KommanderCluster is the Schema for the kommander clusters API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1336]	false
spec		KommanderClusterSpec (see page 1807)	false
status		KommanderClusterStatus (see page 1809)	false

12.1.4.8 KommanderClusterList

KommanderClusterList contains a list of Cluster.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1337]	false
items		[...]KommanderCluster (see page 1807)	true

12.1.4.9 KommanderClusterSpec

KommanderClusterSpec defines the desired state of Cluster.

1336 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1337 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

Field	Description	Scheme	Required
clusterRef		ClusterReference (see page 1804)	false
clusterTunnelConnectorRef	ClusterTunnelConnectorRef is a reference to TunnelConnector that should be used for connecting to cluster.	corev1.LocalObjectReference[1338]	false
clusterTunnelProxyConnectorRef	ClusterTunnelProxyConnectorRef is a reference to TunnelProxy that should be used for connecting to the cluster.	corev1.LocalObjectReference[1339]	false
ingress	Ingress configures the Kommander managed Ingress. If no value is provided, the default ingress will be provisioned.	IngressSpec (see page 1804)	false
kubeconfigRef		corev1.LocalObjectReference[1340]	false

1338 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1339 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1340 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

Field	Description	Scheme	Required
namespaceRef	NamespaceRef will override the namespace on target cluster for the given Workspace. This is used in special circumstances such as when an essential cluster is attached and thus implicitly demoted from being a management cluster to a managed cluster thus forcing the kommander namespace to be recycled as the underlying workspace namespace.	corev1.LocalObjectReference[1341]	false

12.1.4.10 KommanderClusterStatus

KommanderClusterStatus defines the observed state of Cluster.

Field	Description	Scheme	Required
type	ClusterType represents the type of the cluster. E.g. EKS, GKE, etc.	string	false
conditions		[...] metav1.Condition[1342]	false
dextfaclientRef	DexTFAClientRef holds a reference to a DexClient provisioned for Traefik Forward Auth running on managed cluster.	corev1.ObjectReference[1343]	false

1341 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1342 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#condition-v1-meta
1343 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectreference-v1-core

Field	Description	Scheme	Required
ingress	Ingress holds information about the cluster's Ingress	IngressStatus (see page 1805)	false
kubefedclusterRef	KubefedClusterRef holds a reference to a kubefedcluster in the kubefed system namespace.	corev1.LocalObjectReference[1344]	false
clusterId	KubernetesClusterID is the stable cluster ID of the Kubernetes cluster that this Kommander cluster represents.	string	false
kubernetesVersion	KubernetesVersion is the Kubernetes version of the cluster.	string	false
phase	Phase represents the current phase of cluster actuation. E.g. Pending, Provisioning, Provisioned, Deleting, Failed, etc.	string	false
serviceEndpoints	ServiceEndpoints will be the addresses assigned to the Kubernetes exposed services	object	false

12.1.4.11 License

License is the Schema for the licenses API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1345]	false

1344 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1345 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

Field	Description	Scheme	Required
spec		LicenseSpec (see page 1812)	false
status		LicenseStatus (see page 1813)	false

12.1.4.12 LicenseCondition

Field	Description	Scheme	Required
lastTransitionTime		metav1.Time[1346]	false
message		string	false
reason		string	false
status		string	true
type		string	true

12.1.4.13 LicenseExternalAWS

Field	Description	Scheme	Required
licenseArn	LicenseArn is a fully qualified AWS arn to a license for Kommander.	string	false

1346 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#time-v1-meta

12.1.4.14 LicenseExternalReference

Field	Description	Scheme	Required
awsLicense	AWSLicense holds a single reference to a license in AWS's License Manager	LicenseExternalAWS (see page 1811)	false
type	Type is the source of the external license referenced, i.e. AWS, GCP, Azure	string	true

12.1.4.15 LicenseList

LicenseList contains a list of License.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1347]	false
items		[...]License (see page 1810)	true

12.1.4.16 LicenseSpec

LicenseSpec defines the desired state of License.

Field	Description	Scheme	Required
externalLicenseRef	ExternalLicenseRef holds a reference to an external license	LicenseExternalReference (see page 1812)	false

1347 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

Field	Description	Scheme	Required
licenseRef	LicenseReference holds a single reference to the secret holding the license JWT	corev1.LocalObjectReference[1348]	false

12.1.4.17 LicenseStatus

LicenseStatus defines the observed state of License.

Field	Description	Scheme	Required
clusterCapacity	Maximum number of clusters that the license allows.	int32	true
conditions	Conditions relevant to the license (currently used to track term breaches)	[...]LicenseCondition (see page 1811)	false
coreCapacity	Maximum number of cores that the license allows.	int32	true
customerId	The customer's ID. This is the customer name provided from Salesforce.	string	true
dkpLevel	The DKP license level.	string	true
endDate	End date of the licensing period.	metav1.Time[1349]	false
licenseId	The license's ID as provided from Salesforce.	string	true

1348 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1349 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#time-v1-meta

Field	Description	Scheme	Required
productName	The product name the license is for.	string	true
productSKU	The product SKU the license is for.	string	true
startDate	Start date of the licensing period.	metav1.Time[1350]	false
valid	Indicates whether the license is valid, i.e. the secret containing the JWT exists and the JWT carries a valid D2iQ signature. This does NOT indicate whether the license has expired or terms have been breached.	bool	true
version	The license's version ID as provided when the license was created.	string	true

12.1.4.18 PlacementSelector

PlacementSelector defines the fields to select clusters where to apply the project.

Field	Description	Scheme	Required
clusterSelector		metav1.LabelSelector[1351]	false
clusters		[...]GenericClusterReference (see page 1804)	false

1350 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#time-v1-meta
1351 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#labelselector-v1-meta

12.1.4.19 VirtualGroup

VirtualGroup is the Schema for the virtualgroups API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1352]	false
spec		VirtualGroupSpec (see page 1816)	false

12.1.4.20 VirtualGroupClusterRoleBinding

VirtualGroupClusterRoleBinding is the Schema for the virtualgroupclusterrolebindings API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1353]	false
spec		VirtualGroupClusterRoleBindingSpec (see page 1816)	true

12.1.4.21 VirtualGroupClusterRoleBindingList

VirtualGroupClusterRoleBindingList contains a list of VirtualGroupClusterRoleBinding.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1354]	false
items		[...]VirtualGroupClusterRoleBinding (see page 1815)	true

1352 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1353 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1354 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

12.1.4.22 VirtualGroupClusterRoleBindingSpec

VirtualGroupClusterRoleBindingSpec defines the desired state of VirtualGroupClusterRoleBinding.

Field	Description	Scheme	Required
clusterRoleRef		corev1.LocalObjectReference[1355]	true
placement		PlacementSelector (see page 1814)	false
virtualGroupRef		corev1.LocalObjectReference[1356]	true

12.1.4.23 VirtualGroupList

VirtualGroupList contains a list of VirtualGroup.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1357]	false
items		[...]VirtualGroup (see page 1815)	true

12.1.4.24 VirtualGroupSpec

VirtualGroupSpec defines the desired state of VirtualGroup.

Field	Description	Scheme	Required
subjects		[...]rbacv1.Subject[1358]	false

1355 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1356 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1357 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1358 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#subject-v1-rbac-authorization-k8s-io

12.1.5 workspaces.kommander.mesosphere.io/v1alpha1

12.1.5.1 KommanderProjectRole

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1359]	false
spec		KommanderProjectRoleSpec (see page 1817)	false
status		KommanderProjectRoleStatus (see page 1818)	false

12.1.5.2 KommanderProjectRoleList

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1360]	false
items		[...]KommanderProjectRole (see page 1817)	true

12.1.5.3 KommanderProjectRoleSpec

Field	Description	Scheme	Required
projectObjectVerbs		[...]string	false
rules		[...] rbacv1.PolicyRule[1361]	false

1359 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1360 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1361 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#policyrule-v1-rbac-authorization-k8s-io

12.1.5.4 KommanderProjectRoleStatus

Field	Description	Scheme	Required
roleInProjectRef		corev1.LocalObjectReference[1362]	false
roleInWorkspaceRef		corev1.LocalObjectReference[1363]	false

12.1.5.5 KommanderWorkspaceRole

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1364]	false
spec		KommanderWorkspaceRoleSpec (see page 1819)	false
status		KommanderWorkspaceRoleStatus (see page 1819)	false

12.1.5.6 KommanderWorkspaceRoleList

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1365]	false
items		[...]KommanderWorkspaceRole (see page 1818)	true

1362 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1363 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1364 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1365 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

12.1.5.7 KommanderWorkspaceRoleSpec

Field	Description	Scheme	Required
rules		[...] rbacv1.PolicyRule[1366]	false
workspaceObjectVerbs		[...]string	false

12.1.5.8 KommanderWorkspaceRoleStatus

Field	Description	Scheme	Required
clusterRoleRef		corev1.LocalObjectReference[1367]	false
roleInWorkspaceRef		corev1.LocalObjectReference[1368]	false

12.1.5.9 Project

Project is a logical top-level container for a set of Kommander resources.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1369]	false
spec		ProjectSpec (see page 1822)	false
status		ProjectStatus (see page 1822)	false

1366 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#policyrule-v1-rbac-authorization-k8s-io
1367 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1368 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1369 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

12.1.5.10 ProjectCondition

Field	Description	Scheme	Required
lastTransitionTime	Last time the condition transitioned from one status to another.	metav1.Time[1370]	false
message	A human readable message indicating details about the transition.	string	false
reason	The reason for the condition's last transition.	string	false
status	Status of the condition, one of True, False, Unknown.	string	true
type	Type of project condition.	ProjectConditionType	true

12.1.5.11 ProjectList

ProjectList is a list of Project objects.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1371]	false
items		[...]Project (see page 1819)	true

1370 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#time-v1-meta
1371 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

12.1.5.12 ProjectRole

ProjectRole is the Schema for the workspaces ProjectRole API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1372]	false
spec		ProjectRoleSpec (see page 1821)	false
status		ProjectRoleStatus (see page 1822)	false

12.1.5.13 ProjectRoleList

ProjectRoleList contains a list of ProjectRole.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1373]	false
items		[...]ProjectRole (see page 1821)	true

12.1.5.14 ProjectRoleSpec

Field	Description	Scheme	Required
rules		[...] rbacv1.PolicyRule[1374]	false

1372 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1373 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1374 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#policyrule-v1-rbac-authorization-k8s-io

12.1.5.15 ProjectRoleStatus

Field	Description	Scheme	Required
federatedRoleRef		corev1.LocalObjectReference[1375]	false

12.1.5.16 ProjectSpec

ProjectSpec describes the attributes on a Project.

Field	Description	Scheme	Required
namespaceName	NamespaceName specifies the optional namespace name to use for the project. This field is immutable, only settable on create.	string	false
placement		v1beta1.PlacementSelector[1376]	false
workspaceRef		corev1.LocalObjectReference[1377]	false

12.1.5.17 ProjectStatus

ProjectStatus is information about the current status of a Project.

1375 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1376 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/110854693/kommander.mesosphere.io+v1beta1#PlacementSelector
1377 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

Field	Description	Scheme	Required
conditions	Represents the latest available observations of a project's current state.	[...]ProjectCondition (see page 1820)	false
namespaceRef		corev1.LocalObjectReference[1378]	false

12.1.5.18 VirtualGroupKommanderClusterRoleBinding

VirtualGroupKommanderClusterRoleBinding is the Schema for the VirtualGroupWorkspaceRoleBinding API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1379]	false
spec		VirtualGroupKommanderClusterRoleBindingSpec (see page 1824)	true
status		VirtualGroupKommanderClusterRoleBindingStatus (see page 1824)	false

12.1.5.19 VirtualGroupKommanderClusterRoleBindingList

VirtualGroupKommanderClusterRoleBindingList contains a list of VirtualGroupKommanderClusterRoleBinding.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1380]	false

1378 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1379 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1380 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

Field	Description	Scheme	Required
items		[...]VirtualGroupKommanderClusterRoleBinding (see page 1823)	true

12.1.5.20 VirtualGroupKommanderClusterRoleBindingSpec

Field	Description	Scheme	Required
clusterRoleRef		corev1.LocalObjectReference[1381]	true
virtualGroupRef		corev1.LocalObjectReference[1382]	true

12.1.5.21 VirtualGroupKommanderClusterRoleBindingStatus

Field	Description	Scheme	Required
clusterRoleBindingRef		corev1.LocalObjectReference[1383]	false

12.1.5.22 VirtualGroupKommanderProjectRoleBinding

VirtualGroupKommanderProjectRoleBinding is the Schema to be used in the API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1384]	false

1381 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1382 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1383 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1384 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

Field	Description	Scheme	Required
spec		VirtualGroupKommanderProjectRoleBindingSpec (see page 1825)	true
status		VirtualGroupKommanderProjectRoleBindingStatus (see page 1826)	false

12.1.5.23 VirtualGroupKommanderProjectRoleBindingList

VirtualGroupKommanderProjectRoleBindingList contains a list of VirtualGroupKommanderProjectRoleBinding.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1385]	false
items		[...]VirtualGroupKommanderProjectRoleBinding (see page 1824)	true

12.1.5.24 VirtualGroupKommanderProjectRoleBindingSpec

Field	Description	Scheme	Required
kommanderProjectRoleRef		corev1.LocalObjectReference[1386]	true
virtualGroupRef		corev1.LocalObjectReference[1387]	true

1385 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1386 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1387 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

12.1.5.25 VirtualGroupKommanderProjectRoleBindingStatus

Field	Description	Scheme	Required
roleBindingInProjectRef		corev1.LocalObjectReference[1388]	false
roleBindingInWorkspaceRef		corev1.LocalObjectReference[1389]	false

12.1.5.26 VirtualGroupKommanderWorkspaceRoleBinding

VirtualGroupKommanderWorkspaceRoleBinding is the Schema to be used in the API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1390]	false
spec		VirtualGroupKommanderWorkspaceRoleBindingSpec (see page 1827)	true
status		VirtualGroupKommanderWorkspaceRoleBindingStatus (see page 1827)	false

12.1.5.27 VirtualGroupKommanderWorkspaceRoleBindingList

VirtualGroupKommanderWorkspaceRoleBindingList contains a list of VirtualGroupKommanderWorkspaceRoleBinding.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1391]	false

1388 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1389 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1390 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1391 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta

Field	Description	Scheme	Required
items		[...]VirtualGroupKommanderWorkspaceRoleBinding (see page 1826)	true

12.1.5.28 VirtualGroupKommanderWorkspaceRoleBindingSpec

Field	Description	Scheme	Required
kommanderWorkspaceRoleRef		corev1.LocalObjectReference[1392]	true
virtualGroupRef		corev1.LocalObjectReference[1393]	true

12.1.5.29 VirtualGroupKommanderWorkspaceRoleBindingStatus

Field	Description	Scheme	Required
clusterRoleBindingRef		corev1.LocalObjectReference[1394]	false
roleBindingInWorkspaceRef		corev1.LocalObjectReference[1395]	false

12.1.5.30 VirtualGroupProjectRoleBinding

VirtualGroupProjectRoleBinding is the Schema for the VirtualGroupProjectRoleBinding API.

1392 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1393 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1394 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1395 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1396]	false
spec		VirtualGroupProjectRoleBindingSpec (see page 1828)	true
status		VirtualGroupProjectRoleBindingStatus (see page 1829)	false

12.1.5.31 VirtualGroupProjectRoleBindingList

VirtualGroupProjectRoleBindingList contains a list of VirtualGroupProjectRoleBinding.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1397]	false
items		[...]VirtualGroupProjectRoleBinding (see page 1827)	true

12.1.5.32 VirtualGroupProjectRoleBindingSpec

Field	Description	Scheme	Required
projectRoleRef		corev1.LocalObjectReference[1398]	false
virtualGroupRef		corev1.LocalObjectReference[1399]	true

1396 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1397 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1398 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1399 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

Field	Description	Scheme	Required
workspaceRoleRef	WorkspaceRoleRef maybe a LocalObjectReference but the WorkspaceRole is not created in project namespace but in Workspace namespace. `Local` in LocalObjectReference means `Local to project's workspace` since there can only be one workspace the project is in.	corev1.LocalObjectReference[1400]	false

12.1.5.33 VirtualGroupProjectRoleBindingStatus

Field	Description	Scheme	Required
federatedRoleBindingRef		corev1.LocalObjectReference[1401]	false

12.1.5.34 VirtualGroupWorkspaceRoleBinding

VirtualGroupWorkspaceRoleBinding is the Schema for the VirtualGroupWorkspaceRoleBinding API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1402]	false
spec		VirtualGroupWorkspaceRoleBindingSpec (see page 1830)	true

1400 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1401 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1402 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

Field	Description	Scheme	Required
status		VirtualGroupWorkspaceRoleBindingStatus (see page 1831)	false

12.1.5.35 VirtualGroupWorkspaceRoleBindingList

VirtualGroupWorkspaceRoleBindingList contains a list of VirtualGroupWorkspaceRoleBinding.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1403]	false
items		[...]VirtualGroupWorkspaceRoleBinding (see page 1829)	true

12.1.5.36 VirtualGroupWorkspaceRoleBindingSpec

Field	Description	Scheme	Required
placement		v1beta1.PlacementSelector[1404]	false
virtualGroupRef		corev1.LocalObjectReference[1405]	true
workspaceRoleRef		corev1.LocalObjectReference[1406]	true

1403 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1404 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/110854693/kommander.mesosphere.io+v1beta1#PlacementSelector
1405 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1406 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

12.1.5.37 VirtualGroupWorkspaceRoleBindingStatus

Field	Description	Scheme	Required
federatedClusterRoleBindingRef		corev1.LocalObjectReference[1407]	false

12.1.5.38 Workspace

Workspace is the Schema for the workspaces API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1408]	false
spec		WorkspaceSpec (see page 1834)	false
status		WorkspaceStatus (see page 1834)	false

12.1.5.39 WorkspaceCondition

Field	Description	Scheme	Required
lastTransitionTime	Last time the condition transitioned from one status to another.	metav1.Time[1409]	false
message	A human readable message indicating details about the transition.	string	false

1407 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core
1408 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta
1409 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#time-v1-meta

Field	Description	Scheme	Required
reason	The reason for the condition's last transition.	string	false
status	Status of the condition, one of True, False, Unknown.	string	true
type	Type of workspace condition.	string	true

12.1.5.40 WorkspaceList

WorkspaceList contains a list of Workspace.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1410]	false
items		[...]Workspace (see page 1831)	true

12.1.5.41 WorkspaceRole

WorkspaceRole is the Schema for the workspaces API.

Field	Description	Scheme	Required
metadata		metav1.ObjectMeta[1411]	false
spec		WorkspaceRoleSpec (see page 1833)	false
status		WorkspaceRoleStatus (see page 1833)	false

1410 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1411 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#objectmeta-v1-meta

12.1.5.42 WorkspaceRoleList

WorkspaceRoleList contains a list of WorkspaceRole.

Field	Description	Scheme	Required
metadata		metav1.ListMeta[1412]	false
items		[...]WorkspaceRole (see page 1832)	true

12.1.5.43 WorkspaceRoleSpec

Field	Description	Scheme	Required
aggregationRule		rbacv1.AggregationRule[1413]	false
rules		[...] rbacv1.PolicyRule[1414]	false

12.1.5.44 WorkspaceRoleStatus

Field	Description	Scheme	Required
federatedClusterRoleRef		corev1.LocalObjectReference[1415]	false

1412 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#listmeta-v1-meta
1413 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#aggregationrule-v1-rbac-authorization-k8s-io
1414 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#policyrule-v1-rbac-authorization-k8s-io
1415 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

12.1.5.45 WorkspaceSpec

Field	Description	Scheme	Required
clusterLabels		object	false
namespaceName	NamespaceName specifies the optional namespace name to use for the workspace. This field is immutable, only settable on create.	string	false
version	Version stores the desired version of kommander, which should match the current version of the kommander operator.	string	false

12.1.5.46 WorkspaceStatus

Field	Description	Scheme	Required
conditions	Represents the latest available observations of a workspace's current state.	[...]WorkspaceCondition (see page 1831)	false
namespaceRef		corev1.LocalObjectReference[1416]	false
version	Version stores the current version of kommander.	string	false

1416 https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#localobjectreference-v1-core

12.2 CLI Commands

12.2.1 CLI Commands for DKP

The table of contents on the left lists the command groupings. Inside each section, you will find the individual commands for most actions. For KIB CLI, refer to the documentation section for Konvoy Image Builder CLI (see page 1683)commands.

12.2.2 dkp attach

Attach one of [cluster]

12.2.2.1 Options

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
```

```
  -h, --help                     help for attach
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.2.2 SEE ALSO

- dkp attach cluster (see page 1836) - Attach a cluster

12.2.2.3 dkp attach cluster

Attach a cluster

```
dkp attach cluster -n NAME --attached-kubeconfig FILENAME [flags]
```

12.2.2.3.1 Options

```
      --attached-kubeconfig string   Path of the kubeconfig file of the cluster to be
attached
  -h, --help                         help for cluster
  -n, --name string                  Desired name of the attached cluster
  -w, --workspace string             Name of the workspace of the attached cluster
```

12.2.2.3.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.2.3.3 SEE ALSO

- dkp attach (see page 1835) - Attach one of [cluster]

12.2.3 dkp check

Check, one of [cluster]

12.2.3.1 Options

```
-h, --help          help for check
-v, --verbose int   Output verbosity
```

12.2.3.2 SEE ALSO

- dkp check cluster (see page 1837) - Check a cluster, one of [fips]

12.2.3.3 dkp check cluster

Check a cluster, one of [fips]

12.2.3.3.1 Options

```
-h, --help   help for cluster
```

12.2.3.3.2 Options inherited from parent commands

```
-v, --verbose int   Output verbosity
```

12.2.3.3.3 SEE ALSO

- dkp check (see page 1837) - Check, one of [cluster]
- dkp check cluster fips (see page 1837) - Validate the components in your cluster are FIPS compliant

12.2.3.3.4 dkp check cluster fips

Validate the components in your cluster are FIPS compliant

12.2.3.3.4.1 Synopsis

The check cluster fips command is used to validate that specific components and services are FIPS compliant by checking the signatures of the files against a signed signature file, and checking that services are using the certified algorithms.

Examples:

```
To use the built-in signature files for supported operating systems:

dkp check cluster fips

To use a custom signature file, named "manifest-rhel-84.json.asc":

dkp check cluster fips \
    --signature-file manifest-rhel-84.json.asc \
    --signature-configmap myconfigmap

The file will be copied to the ConfigMap. To use an existing ConfigMap:

dkp check cluster fips \
    --signature-configmap myconfigmap

The validation will be re-checked against the existing signature data.
```

```
dkp check cluster fips [flags]
```

12.2.3.3.4.2 Options

```
  -h, --help                          help for fips
      --kubeconfig string             Path to the kubeconfig file for the fips
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
      --output-configmap string       ConfigMap to store result of the fips check.
(default "check-cluster-fips-output") (DEPRECATED: This flag will be removed in a
future release.)
      --signature-configmap string    ConfigMap with fips signature data to verify.
      --signature-file string         File containing fips signature data.
      --timeout duration              The length of time to wait before giving up.
Zero means wait forever (e.g. 300s, 30m, 3h). (default 10m0s)
```

12.2.3.3.4.3 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.3.3.4.4 SEE ALSO

- dkp check cluster (see page 1837) - Check a cluster, one of [fips]

12.2.4 dkp cluster

Get cluster information

12.2.4.1 Options

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for cluster
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.4.2 SEE ALSO

- dkp cluster type (see page 1839) - Retrieve cluster type

12.2.4.3 dkp cluster type

Retrieve cluster type

```
dkp cluster type [flags]
```

12.2.4.3.1 Options

```
  -h, --help   help for type
```

12.2.4.3.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.4.3.3 SEE ALSO

- dkp cluster (see page 1839) - Get cluster information

12.2.5 dkp completion

Generate the autocompletion script for the specified shell

12.2.5.1 Synopsis

Generate the autocompletion script for dkp for the specified shell. See each sub-command's help for details on how to use the generated script.

12.2.5.2 Options

```
  -h, --help   help for completion
```

12.2.5.3 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.5.4 SEE ALSO

- dkp completion bash (see page 1841) - Generate the autocompletion script for bash
- dkp completion fish (see page 1841) - Generate the autocompletion script for fish
- dkp completion powershell (see page 1844) - Generate the autocompletion script for powershell
- dkp completion zsh (see page 1843) - Generate the autocompletion script for zsh

12.2.5.5 dkp completion fish

Generate the autocompletion script for fish

12.2.5.5.1 Synopsis

Generate the autocompletion script for the fish shell.

To load completions in your current shell session:

```
dkp completion fish | source
```

To load completions for every new session, execute once:

```
dkp completion fish > ~/.config/fish/completions/dkp.fish
```

You will need to start a new shell for this setup to take effect.

```
dkp completion fish [flags]
```

12.2.5.5.2 Options

```
  -h, --help              help for fish
      --no-descriptions   disable completion descriptions
```

12.2.5.5.3 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.5.5.4 SEE ALSO

- dkp completion (see page 1840) - Generate the autocompletion script for the specified shell

12.2.5.6 dkp completion bash

Generate the autocompletion script for bash

12.2.5.6.1 Synopsis

Generate the autocompletion script for the bash shell.

This script depends on the 'bash-completion' package. If it is not installed already, you can install it via your OS's package manager.

To load completions in your current shell session:

```
source <(dkp completion bash)
```

To load completions for every new session, execute once:

12.2.5.6.1.1 Linux:

```
dkp completion bash > /etc/bash_completion.d/dkp
```

12.2.5.6.1.2 macOS:

```
dkp completion bash > $(brew --prefix)/etc/bash_completion.d/dkp
```

You will need to start a new shell for this setup to take effect.

```
dkp completion bash
```

12.2.5.6.2 Options

```
  -h, --help              help for bash
      --no-descriptions   disable completion descriptions
```

12.2.5.6.3 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.5.6.4 SEE ALSO

- dkp completion (see page 1840) - Generate the autocompletion script for the specified shell

12.2.5.7 dkp completion zsh

Generate the autocompletion script for zsh

12.2.5.7.1 Synopsis

Generate the autocompletion script for the zsh shell.

If shell completion is not already enabled in your environment you will need to enable it. You can execute the following once:

```
echo "autoload -U compinit; compinit" >> ~/.zshrc
```

To load completions in your current shell session:

```
source <(dkp completion zsh)
```

To load completions for every new session, execute once:

12.2.5.7.1.1 Linux:

```
dkp completion zsh > "${fpath[1]}/_dkp"
```

12.2.5.7.1.2 macOS:

```
dkp completion zsh > $(brew --prefix)/share/zsh/site-functions/_dkp
```

You will need to start a new shell for this setup to take effect.

```
dkp completion zsh [flags]
```

12.2.5.7.2 Options

```
  -h, --help              help for zsh
      --no-descriptions   disable completion descriptions
```

12.2.5.7.3 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.5.7.4 SEE ALSO

- dkp completion (see page 1840) - Generate the autocompletion script for the specified shell

12.2.5.8 dkp completion powershell

Generate the autocompletion script for powershell

12.2.5.8.1 Synopsis

Generate the autocompletion script for powershell.

To load completions in your current shell session:

```
dkp completion powershell | Out-String | Invoke-Expression
```

To load completions for every new session, add the output of the above command to your powershell profile.

```
dkp completion powershell [flags]
```

12.2.5.8.2 Options

```
  -h, --help              help for powershell
      --no-descriptions   disable completion descriptions
```

12.2.5.8.3 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.5.8.4 SEE ALSO

- dkp completion (see page 1840) - Generate the autocompletion script for the specified shell

12.2.6 dkp config

Manage DKP Kommander's configuration

12.2.6.1 Options

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for config
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.6.2 SEE ALSO

- dkp config get (see page 1845) - Retrieve DKP Kommander's configuration
- dkp config set (see page 1846) - Modify DKP Kommander's configuration

12.2.6.3 dkp config get

Retrieve DKP Kommander's configuration

12.2.6.3.1 Options

```
  -h, --help   help for get
```

12.2.6.3.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.6.3.3 SEE ALSO

- dkp config (see page 1845) - Manage DKP Kommander's configuration
- dkp config get default-workspace (see page 1846) - Displays the name of the configured default Workspace

12.2.6.3.4 dkp config get default-workspace

Displays the name of the configured default Workspace

```
dkp config get default-workspace [flags]
```

12.2.6.3.4.1 Options

```
  -h, --help   help for default-workspace
```

12.2.6.3.4.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.6.3.4.3 SEE ALSO

- dkp config get (see page 1845) - Retrieve DKP Kommander's configuration

12.2.6.4 dkp config set

Modify DKP Kommander's configuration

12.2.6.4.1 Options

```
  -h, --help   help for set
```

12.2.6.4.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.6.4.3 SEE ALSO

- dkp config (see page 1845) - Manage DKP Kommander's configuration
- dkp config set default-workspace (see page 1847) - Set the name of the default Workspace to use in all commands when none is provided

12.2.6.4.4 dkp config set default-workspace

Set the name of the default Workspace to use in all commands when none is provided

```
dkp config set default-workspace [WORKSPACE] [flags]
```

12.2.6.4.4.1 Options

```
  -h, --help   help for default-workspace
```

12.2.6.4.4.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
```

```
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.6.4.4.3 SEE ALSO

- dkp config set (see page 1846) - Modify DKP Kommander's configuration

12.2.7 dkp create

Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]

12.2.7.1 Options

```
  -h, --help   help for create
```

12.2.7.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.3 SEE ALSO

- dkp create appdeployment (see page 1849) - Create an AppDeployment
- dkp create bootstrap (see page 1852) - Create bootstrap cluster
- dkp create capi-components (see page 1850) - Create the CAPI components in the cluster
- dkp create chart-bundle (see page 1852) - Create charts bundle based on a catalog applications git repository
- dkp create cluster (see page 1853) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]
- dkp create image-bundle (see page 1851) - Create an image bundle
- dkp create nodepool (see page 1875) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]
- dkp create workspace (see page 1848) - Create a Workspace

12.2.7.4 dkp create workspace

Create a Workspace

```
dkp create workspace WORKSPACE_NAME [flags]
```

12.2.7.4.1 Options

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
      --dry-run                  Export in YAML format to stdout
  -h, --help                     help for workspace
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
  -n, --namespace string         Name of the Namespace to create for the workspace
  -o, --output string            Output format. One of: yaml|json (default "yaml")
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.7.4.2 SEE ALSO

- dkp create (see page 1848) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]

12.2.7.5 dkp create appdeployment

Create an AppDeployment

12.2.7.5.1 Synopsis

Creates an AppDeployment in a workspace or project.

When [--clusters C1,C2..] is not specified, it targets all existing clusters in the specified workspace or project.

```
dkp create appdeployment APPDEPLOYMENT_NAME --app NAME [flags]
```

12.2.7.5.2 Options

```
      --add-cluster-config-overrides strings   Comma-separated list of mappings of
kommanderCluster name to cluster configuration override ConfigMap name to apply to
the targeting clusters. (e.g., cluster-1:override-1-cm,cluster-2:override-2-cm)(only
valid for dkp clusters version 2.3.x and up) (default [])
  -a, --app string                             Name of the App to deploy
```

```
      --clusters strings                        List of names of kommanderClusters to
select for the command. (e.g., cluster-1,cluster-2)(only valid for dkp clusters
version 2.3.x and up) (default [])
      --config string                           Config file to use (default "/home/
runner/.kommander/config")
  -c, --config-overrides string                 Name of the ConfigMap used to override
default configuration of the App
      --context string                          The name of the kubeconfig context to
use
      --dry-run                                 Export in YAML format to stdout
  -h, --help                                    help for appdeployment
      --kubeconfig string                       Path to the kubeconfig file to use for
 CLI requests.
  -o, --output string                           Output format. One of: yaml|json
(default "yaml")
  -p, --project string                          Name of the project to create the
AppDeployment in. Requires workspace flag (workspace that the project belongs to).
      --request-timeout string                  The length of time to wait before
giving up on a single server request. Non-zero values should contain a corresponding
time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int                             Output verbosity
  -w, --workspace string                        Name of the workspace to create the
AppDeployment in
```

12.2.7.5.3 SEE ALSO

- dkp create (see page 1848) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]

12.2.7.6 dkp create capi-components

Create the CAPI components in the cluster

```
dkp create capi-components [flags]
```

12.2.7.6.1 Options

```
      --aws-service-endpoints string     Custom AWS service endpoints in a semi-colon
separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=${URL};$
{SigningRegion2}...
  -h, --help                             help for capi-components
      --http-proxy string                HTTP proxy for CAPI controllers
      --https-proxy string               HTTPS proxy for CAPI controllers
      --kubeconfig string                Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
```

```
      --no-proxy strings                No Proxy list for CAPI controllers (default
 [])
      --timeout duration                The length of time to wait before giving up.
Zero means wait forever (e.g. 300s, 30m, 3h). (default 10m0s)
  -v, --verbose int                     Output verbosity
      --wait                            If true, wait for operations to complete
before returning. (default true)
      --with-aws-bootstrap-credentials   Set true to use AWS bootstrap credentials
from your environment. When false, the instance profile of the EC2 instance where the
CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials   Set true to use GCP bootstrap credentials
from your environment. When false, the service account of the VM instance where the
CAPG controller is scheduled on will be used instead.
```

12.2.7.6.2 SEE ALSO

- dkp create (see page 1848) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]

12.2.7.7 dkp create image-bundle

Create an image bundle

```
dkp create image-bundle [flags]
```

12.2.7.7.1 Options

```
  -h, --help                          help for image-bundle
      --image-pull-concurrency int    Image pull concurrency (default 1)
      --images-file string            File containing list of images to create bundle
from, either as YAML configuration or a simple list of images
      --output-file string            Output file to write image bundle to (default
"images.tar")
      --overwrite                     Overwrite image bundle file if it already exists
      --platform platformSlice        platforms to download images (required format:
<os>/<arch>[/<variant>]) (default [linux/amd64])
  -v, --verbose int                   Output verbosity
```

12.2.7.7.2 SEE ALSO

- dkp create (see page 1848) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]

12.2.7.8 dkp create bootstrap

Create bootstrap cluster

```
dkp create bootstrap [flags]
```

12.2.7.8.1 Options

```
      --aws-service-endpoints string     Custom AWS service endpoints in a semi-colon
separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=${URL};$
{SigningRegion2}...
  -h, --help                             help for bootstrap
      --http-proxy string                HTTP proxy for CAPI controllers
      --https-proxy string               HTTPS proxy for CAPI controllers
      --kind-cluster-image string        Kind node image for the bootstrap cluster (d
efault "mesosphere/konvoy-bootstrap:v2.8.1")
      --kubeconfig string                Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --no-proxy strings                 No Proxy list for CAPI controllers (default
 [])
      --timeout duration                 The length of time to wait before giving up.
Zero means wait forever (e.g. 300s, 30m, 3h). (default 10m0s)
  -v, --verbose int                      Output verbosity
      --wait                             If true, wait for operations to complete
before returning. (default true)
      --with-aws-bootstrap-credentials   Set true to use AWS bootstrap credentials
from your environment. When false, the instance profile of the EC2 instance where the
CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials   Set true to use GCP bootstrap credentials
from your environment. When false, the service account of the VM instance where the
CAPG controller is scheduled on will be used instead.
```

12.2.7.8.2 SEE ALSO

- dkp create (see page 1848) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]

12.2.7.9 dkp create chart-bundle

Create charts bundle based on a catalog applications git repository

```
dkp create chart-bundle [flags]
```

12.2.7.9.1 Options

```
      --catalog-repository string         Git repository containing catalog
application definitions
      --config string                     Config file to use (default "/home/
runner/.kommander/config")
      --context string                    The name of the kubeconfig context to use
      --dry-run                           Export in YAML format to stdout
      --extra-charts strings              Extra charts to include in the bundle, in
the format <repo-url1>|<chart-name1>,<repo-url2>|<chart-name2>:<chart-version2>,...
(default [])
  -h, --help                              help for chart-bundle
      --kommander-charts-version string   Kommander helm charts version to download.
Default: download all available versions
      --kubeconfig string                 Path to the kubeconfig file to use for CLI
requests.
  -o, --output string                     File path to write charts bundle to
(default "charts-bundle.tar.gz")
      --request-timeout string            The length of time to wait before giving up
on a single server request. Non-zero values should contain a corresponding time unit
(e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
      --skip-charts strings               Charts to not to include in the bundle, in
the format <chart-name1>,<chart-name2>:<chart-version2>,... (default [])
  -v, --verbose int                       Output verbosity
```

12.2.7.9.2 SEE ALSO

- dkp create (see page 1848) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]

12.2.7.10 dkp create cluster

Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.10.1 Options

```
  -h, --help          help for cluster
  -v, --verbose int   Output verbosity
```

12.2.7.10.2 SEE ALSO

- dkp create (see page 1848) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]

- dkp create cluster aks (see page 1864) - Create a Konvoy cluster in AKS
- dkp create cluster aws (see page 1856) - Create a Konvoy cluster in AWS
- dkp create cluster azure (see page 1859) - Create a Konvoy cluster in Azure
- dkp create cluster eks (see page 1854) - Create a Konvoy cluster in EKS
- dkp create cluster gcp (see page 1869) - Create a Konvoy cluster in GCP
- dkp create cluster preprovisioned (see page 1866) - Create a Konvoy cluster on pre-provisioned infrastructure
- dkp create cluster vcd (see page 1872) - Create a Konvoy cluster in VMware Cloud Director
- dkp create cluster vsphere (see page 1861) - Create a Konvoy cluster in vSphere

12.2.7.10.3 dkp create cluster eks

Create a Konvoy cluster in EKS

```
dkp create cluster eks [flags]
```

12.2.7.10.3.1 Options

```
      --additional-security-group-ids strings   A comma separated list of existing
security group IDs to use for machines in addition to those created automatically (de
fault [])
      --additional-tags stringToString          Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys             If true, ignore any errors in
templates when a field or map key is missing in the template. Only applies to golang
and jsonpath output formats. (default true)
      --aws-service-endpoints string            Custom AWS service endpoints in a
semi-colon separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=$
{URL};${SigningRegion2}...
  -c, --cluster-name name                       Name used to prefix the cluster and
all the created resources.
      --dry-run                                 Only print the objects that would be
created, without creating them.
  -h, --help                                    help for eks
      --http-proxy string                       HTTP proxy for CAPI controllers
      --https-proxy string                      HTTPS proxy for CAPI controllers
      --kind-cluster-image string               Kind node image for the bootstrap
cluster (default "mesosphere/konvoy-bootstrap:v2.8.1")
      --kubeconfig string                       Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply.
      --kubernetes-pod-network-cidr cidr        The Kubernetes Pod network CIDR to
use in the cluster (default 192.168.0.0/16)
      --kubernetes-service-cidr cidr            The Kubernetes Service CIDR to use in
the cluster (default 10.96.0.0/12)
      --kubernetes-version string               Kubernetes version (default "1.27.9")
```

```
  -n, --namespace string                          If present, the namespace scope for
this CLI request. (default "default")
      --no-proxy strings                          No Proxy list for CAPI controllers (d
efault [])
  -o, --output string                             Output format. One of: (json, yaml,
name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-
json, jsonpath-file).
      --output-directory string                   Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --region string                             AWS region to deploy cluster to
(default "us-west-2")
      --registry-mirror-cacert file               CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string           Password to authenticate to the
registry mirror with
      --registry-mirror-url url                   URL of a container registry to use as
a mirror in the cluster
      --registry-mirror-username string           Username to authenticate to the
registry mirror with
      --show-managed-fields                       If true, keep the managedFields when
printing objects in JSON or YAML format.
      --subnet-ids strings                        A comma separated list of existing
subnet IDs to use for the kube-apiserver ELB and all control-plane and worker nodes (d
efault [])
      --template string                           Template string or path to template
file to use when -o=go-template, -o=go-template-file. The template format is golang
templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                          The length of time to wait before
giving up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --vpc-id string                             Existing VPC ID to use for the
cluster
      --wait                                      If true, wait for operations to
complete before returning. (default true)
      --with-aws-bootstrap-credentials            Set true to use AWS bootstrap
credentials from your environment. When false, the instance profile of the EC2
instance where the CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials            Set true to use GCP bootstrap
credentials from your environment. When false, the service account of the VM instance
where the CAPG controller is scheduled on will be used instead.
      --worker-availability-zone string           The AvailabilityZone in the region to
deploy the worker nodes to, if not set a random one will be selected (ex. us-west-2a)
      --worker-iam-instance-profile string        Name of the IAM instance profile to
assign to worker machines. (default "nodes.cluster-api-provider-aws.sigs.k8s.io")
      --worker-instance-type string               Worker machine instance type (default
 "m5.2xlarge")
      --worker-replicas int                       Number of workers (default 4)
```

12.2.7.10.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.10.3.3 SEE ALSO

- dkp create cluster (see page 1853) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.10.4 dkp create cluster aws

Create a Konvoy cluster in AWS

```
dkp create cluster aws [flags]
```

12.2.7.10.4.1 Options

```
      --additional-security-group-ids strings       A comma separated list of
existing security group IDs to use for machines in addition to those created
automatically (default [])
      --additional-tags stringToString              Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys                 If true, ignore any errors in
templates when a field or map key is missing in the template. Only applies to golang
and jsonpath output formats. (default true)
      --ami string                                  AMI ID to use for machines
      --ami-base-os string                          Base OS used in search of AMIs.
Examples: 'centos-7', 'ubuntu-18.04', 'ubuntu-20.04'
      --ami-format string                           Query string used in search of
AMIs. Example: When --ami-base-os='rhel8.4', then the string 'prefix-{{.BaseOS}}-?
{{.K8sVersion}}-*' matches any AMIs with the Name 'prefix-rhel8.4-1.28.7
      --ami-owner string                            ID of AWS account used in search
of AMIs
      --aws-service-endpoints string                Custom AWS service endpoints in a
semi-colon separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=$
{URL};${SigningRegion2}...
      --certificate-renew-interval int              The interval number of days
Kubernetes managed PKI certificates are renewed. For example, an Interval value of 30
 means the certificates will be refreshed every 30 days. A value of 0 disables the
feature. (default 0)
  -c, --cluster-name name                           Name used to prefix the cluster
and all the created resources.
```

```
      --control-plane-http-proxy string             HTTP proxy for control plane
machines
      --control-plane-https-proxy string            HTTPS proxy for control plane
machines
      --control-plane-iam-instance-profile string   Name of the IAM instance profile
to assign to control plane machines. (default "control-plane.cluster-api-provider-
aws.sigs.k8s.io")
      --control-plane-instance-type string          Control Plane machine instance
type (default "m5.xlarge")
      --control-plane-no-proxy strings              No Proxy list for control plane
machines (default [])
      --control-plane-replicas int                  Number of control plane nodes (de
fault 3)
      --dry-run                                     Only print the objects that would
be created, without creating them.
      --etcd-image-repository string                The image repository to use for
 pulling the etcd image
      --etcd-version string                         The version of etcd to use.
      --extra-sans strings                          A comma separated list of
additional Subject Alternative Names for the API Server signing cert (default [])
  -h, --help                                        help for aws
      --http-proxy string                           HTTP proxy for CAPI controllers
      --https-proxy string                          HTTPS proxy for CAPI controllers
      --internal-load-balancer                      Make the control plane load
balancer internal, i.e., reachable only within the VPC.
      --kind-cluster-image string                   Kind node image for the bootstrap
cluster (default "mesosphere/konvoy-bootstrap:v2.8.1")
      --kubeconfig string                           Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply. This flag is
ignored if used with the --self-managed flag.
      --kubernetes-image-repository string          The image repository to use for
 pulling kubernetes images
      --kubernetes-pod-network-cidr cidr            The Kubernetes Pod network CIDR
to use in the cluster (default 192.168.0.0/16)
      --kubernetes-service-cidr cidr                The Kubernetes Service CIDR to
use in the cluster (default 10.96.0.0/12)
      --kubernetes-version string                   Kubernetes version (default
"1.28.7")
  -n, --namespace string                            If present, the namespace scope
for this CLI request. (default "default")
      --no-proxy strings                            No Proxy list for CAPI
controllers (default [])
      --os-hint flatcar                             A hint which will allow the
installer to generate appropriate configurations for a target OS. Presently, only the
hint for flatcar is supported.
  -o, --output string                               Output format. One of: (json,
yaml, name, go-template, go-template-file, template, templatefile, jsonpath,
jsonpath-as-json, jsonpath-file).
      --output-directory string                     Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --region string                               AWS region to deploy cluster to
(default "us-west-2")
```

```
      --registry-mirror-cacert file                 CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string             Password to authenticate to the
registry mirror with
      --registry-mirror-url url                     URL of a container registry to
use as a mirror in the cluster
      --registry-mirror-username string             Username to authenticate to the
registry mirror with
      --self-managed                                When set to true, the required
prerequisites are created before creating the cluster and the resulting cluster has
all necessary components deployed onto itself, so it can manage its own cluster
lifecycle. When set to false, a management cluster is used. (default false)
      --show-managed-fields                         If true, keep the managedFields
when printing objects in JSON or YAML format.
      --ssh-public-key-file string                  Path to the authorized SSH key
for the user
      --ssh-username string                         Name of the user to create on the
instance (default "konvoy")
      --subnet-ids strings                          A comma separated list of
existing subnet IDs to use for the kube-apiserver ELB and all control-plane and
worker nodes (default [])
      --template string                             Template string or path to
template file to use when -o=go-template, -o=go-template-file. The template format is
golang templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                            The length of time to wait before
giving up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --vpc-id string                               Existing VPC ID to use for the
cluster
      --wait                                        If true, wait for operations to
complete before returning. This flag is ignored and will always be 'true' if used
with the --self-managed flag. (default true)
      --with-aws-bootstrap-credentials              Set true to use AWS bootstrap
credentials from your environment. When false, the instance profile of the EC2
instance where the CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials              Set true to use GCP bootstrap
credentials from your environment. When false, the service account of the VM instance
where the CAPG controller is scheduled on will be used instead.
      --worker-availability-zone string             The AvailabilityZone in the
region to deploy the worker nodes to, if not set a random one will be selected (ex.
us-west-2a)
      --worker-http-proxy string                    HTTP proxy for nodes
      --worker-https-proxy string                   HTTPS proxy for nodes
      --worker-iam-instance-profile string          Name of the IAM instance profile
to assign to worker machines. (default "nodes.cluster-api-provider-aws.sigs.k8s.io")
      --worker-instance-type string                 Worker machine instance type
(default "m5.2xlarge")
      --worker-no-proxy strings                     No Proxy list for nodes (default
 [])
      --worker-replicas int                         Number of workers (default 4)
```

12.2.7.10.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.10.4.3 SEE ALSO

- dkp create cluster (see page 1853) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.10.5 dkp create cluster azure

Create a Konvoy cluster in Azure

```
dkp create cluster azure [flags]
```

12.2.7.10.5.1 Options

```
      --additional-tags stringToString       Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys          If true, ignore any errors in templates
when a field or map key is missing in the template. Only applies to golang and
jsonpath output formats. (default true)
      --aws-service-endpoints string         Custom AWS service endpoints in a semi-
colon separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=${URL};$
{SigningRegion2}...
      --certificate-renew-interval int       The interval number of days Kubernetes
managed PKI certificates are renewed. For example, an Interval value of 30 means the
certificates will be refreshed every 30 days. A value of 0 disables the feature. (def
ault 0)
  -c, --cluster-name name                    Name used to prefix the cluster and all
the created resources.
      --compute-gallery-id string            Compute Gallery ID of a custom image,
e.g., '/subscriptions/<subscription id>/resourceGroups/<resource group name>/
providers/Microsoft.Compute/galleries/<gallery name>/images/<image definition name>/
versions/<version id>' (replacing placeholders with the values used when creating the
image)
      --control-plane-http-proxy string      HTTP proxy for control plane machines
      --control-plane-https-proxy string     HTTPS proxy for control plane machines
      --control-plane-machine-size string    Control Plane machine size (default
"Standard_D4s_v3")
      --control-plane-no-proxy strings       No Proxy list for control plane machines
(default [])
      --control-plane-replicas int           Number of control plane nodes (default 3)
```

```
      --dry-run                                Only print the objects that would be
created, without creating them.
      --etcd-image-repository string           The image repository to use for pulling
the etcd image
      --etcd-version string                    The version of etcd to use.
      --extra-sans strings                     A comma separated list of additional
Subject Alternative Names for the API Server signing cert (default [])
  -h, --help                                   help for azure
      --http-proxy string                      HTTP proxy for CAPI controllers
      --https-proxy string                     HTTPS proxy for CAPI controllers
      --kind-cluster-image string              Kind node image for the bootstrap
cluster (default "mesosphere/konvoy-bootstrap:v2.8.1")
      --kubeconfig string                      Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply. This flag is
ignored if used with the --self-managed flag.
      --kubernetes-image-repository string     The image repository to use for pulling
kubernetes images
      --kubernetes-pod-network-cidr cidr       The Kubernetes Pod network CIDR to use
in the cluster (default 192.168.0.0/16)
      --kubernetes-service-cidr cidr           The Kubernetes Service CIDR to use in
the cluster (default 10.96.0.0/12)
      --kubernetes-version string              Kubernetes version (default "1.28.7")
      --location string                        Azure location to deploy cluster to
(default "westus")
  -n, --namespace string                       If present, the namespace scope for this
 CLI request. (default "default")
      --no-proxy strings                       No Proxy list for CAPI controllers (defa
ult [])
  -o, --output string                          Output format. One of: (json, yaml,
name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-
json, jsonpath-file).
      --output-directory string                Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --plan-offer string                      The offer for a Marketplace image or a
custom image sourced from a Marketplace image requiring Plan information.
      --plan-publisher string                  The publisher for a Marketplace image or
a custom image sourced from a Marketplace image requiring Plan information.
      --plan-sku string                        The SKU for a Marketplace image or a
custom image sourced from a Marketplace image requiring Plan information.
      --registry-mirror-cacert file            CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string        Password to authenticate to the registry
mirror with
      --registry-mirror-url url                URL of a container registry to use as a
mirror in the cluster
      --registry-mirror-username string        Username to authenticate to the registry
mirror with
      --self-managed                           When set to true, the required
prerequisites are created before creating the cluster and the resulting cluster has
all necessary components deployed onto itself, so it can manage its own cluster
lifecycle. When set to false, a management cluster is used. (default false)
```

```
      --show-managed-fields                 If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string          Path to the authorized SSH key for the
user
      --ssh-username string                 Name of the user to create on the
instance (default "konvoy")
      --template string                     Template string or path to template file
to use when -o=go-template, -o=go-template-file. The template format is golang
templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                    The length of time to wait before giving
up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                                If true, wait for operations to complete
before returning. This flag is ignored and will always be 'true' if used with the --
self-managed flag. (default true)
      --with-aws-bootstrap-credentials      Set true to use AWS bootstrap
credentials from your environment. When false, the instance profile of the EC2
instance where the CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials      Set true to use GCP bootstrap
credentials from your environment. When false, the service account of the VM instance
where the CAPG controller is scheduled on will be used instead.
      --worker-availability-zone string     The availability zone in the region to
deploy the worker nodes to, if not set a random one will be selected (ex. 1). Not all
locations, including the default 'westus', support setting this flag, see https://
docs.microsoft.com/en-us/azure/availability-zones/az-overview.
      --worker-http-proxy string            HTTP proxy for nodes
      --worker-https-proxy string           HTTPS proxy for nodes
      --worker-machine-size string          Worker machine size (default
"Standard_D8s_v3")
      --worker-no-proxy strings             No Proxy list for nodes (default [])
      --worker-replicas int                 Number of workers (default 4)
```

12.2.7.10.5.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.10.5.3 SEE ALSO

- dkp create cluster (see page 1853) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.10.6 dkp create cluster vsphere

Create a Konvoy cluster in vSphere

```
dkp create cluster vsphere [flags]
```

12.2.7.10.6.1 Options

```
      --allow-missing-template-keys          If true, ignore any errors in templates
when a field or map key is missing in the template. Only applies to golang and
jsonpath output formats. (default true)
      --aws-service-endpoints string         Custom AWS service endpoints in a semi-
colon separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=${URL};$
{SigningRegion2}...
      --certificate-renew-interval int       The interval number of days Kubernetes
managed PKI certificates are renewed. For example, an Interval value of 30 means the
certificates will be refreshed every 30 days. A value of 0 disables the feature. (def
ault 0)
  -c, --cluster-name name                    Name used to prefix the cluster and all
the created resources.
      --control-plane-cpus int               The number of virtual processors in a
control plane machine (default 4)
      --control-plane-disk-size int          The size of a control plane machine's
disk, in GB (default 80)
      --control-plane-endpoint-host string   The control plane endpoint address. To
use an external load balancer, set to its IP or hostname. To use the built-in virtual
IP, set to a static IPv4 address in the Layer 2 network of the control plane
machines. [Not for production use: To use a single-machine control plane, set to the
IP or hostname of the machine.]
      --control-plane-endpoint-port int      The control plane endpoint port. To use
an external load balancer, set to its listening port. (default 6443)
      --control-plane-http-proxy string      HTTP proxy for control plane machines
      --control-plane-https-proxy string     HTTPS proxy for control plane machines
      --control-plane-memory int             The size of a control plane machine's
memory, in GB (default 16)
      --control-plane-no-proxy strings       No Proxy list for control plane machines
(default [])
      --control-plane-replicas int           Number of control plane nodes (default 3)
      --data-center string                   The vSphere datacenter to deploy the
workload cluster on.
      --data-store string                    The vSphere datastore to deploy the
workload cluster on.
      --dry-run                              Only print the objects that would be
created, without creating them.
      --etcd-image-repository string         The image repository to use for pulling
the etcd image
      --etcd-version string                  The version of etcd to use.
      --extra-sans strings                   A comma separated list of additional
Subject Alternative Names for the API Server signing cert (default [])
      --folder string                        The vSphere folder for your VMs. Set to
"" to use the root vSphere folder.
  -h, --help                                 help for vsphere
      --http-proxy string                    HTTP proxy for CAPI controllers
      --https-proxy string                   HTTPS proxy for CAPI controllers
```

```
      --kind-cluster-image string            Kind node image for the bootstrap
cluster (default "mesosphere/konvoy-bootstrap:v2.8.1")
      --kubeconfig string                    Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply. This flag is
ignored if used with the --self-managed flag.
      --kubernetes-image-repository string   The image repository to use for pulling
kubernetes images
      --kubernetes-pod-network-cidr cidr     The Kubernetes Pod network CIDR to use
in the cluster (default 192.168.0.0/16)
      --kubernetes-service-cidr cidr         The Kubernetes Service CIDR to use in
the cluster (default 10.96.0.0/12)
      --kubernetes-version string            Kubernetes version (default "1.28.7")
  -n, --namespace string                     If present, the namespace scope for this
 CLI request. (default "default")
      --network string                       The vSphere network to deploy the
workload cluster on.
      --no-proxy strings                     No Proxy list for CAPI controllers (defa
ult [])
      --os-hint flatcar                      A hint which will allow the installer to
generate appropriate configurations for a target OS. Presently, only the hint for
 flatcar is supported.
  -o, --output string                        Output format. One of: (json, yaml,
name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-
json, jsonpath-file).
      --output-directory string              Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --registry-mirror-cacert file          CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string      Password to authenticate to the registry
mirror with
      --registry-mirror-url url              URL of a container registry to use as a
mirror in the cluster
      --registry-mirror-username string      Username to authenticate to the registry
mirror with
      --resource-pool string                 The vSphere resource pool for the
workload cluster's virtual machines.
      --self-managed                         When set to true, the required
prerequisites are created before creating the cluster and the resulting cluster has
all necessary components deployed onto itself, so it can manage its own cluster
lifecycle. When set to false, a management cluster is used. (default false)
      --server string                        The vCenter server address. Accepted
formats: host, host:port, http[s]://host[:port]. Accepted host formats: IPv4, IPv6,
or DNS name.
      --show-managed-fields                  If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string           Path to the authorized SSH key for the
user
      --ssh-username string                  Name of the user to create on the
instance (default "konvoy")
      --storage-policy string                This is the vSphere storage policy. Set
it to "" if you don't want to use a storage policy.
```

```
      --template string                       Template string or path to template file
to use when -o=go-template, -o=go-template-file. The template format is golang
templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                      The length of time to wait before giving
up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --tls-thumb-print string                sha1 thumbprint of the vcenter
certificate: openssl x509 -sha1 -fingerprint -in ca.crt -noout
      --virtual-ip-interface string           The network interface, e.g, 'eth0' or
'ens5', to use for the built-in virtual IP control plane endpoint. This interface
 must be available on every control plane machine. If the value is empty, the flag
does nothing. If the value is not empty, the built-in virtual IP control plane
endpoint is created, using values from --control-plane-endpoint-host and --control-
plane-endpoint-port.
      --vm-template string                    The virtual machine template to use for
 the workload cluster's virtual machines.
      --wait                                  If true, wait for operations to complete
before returning. This flag is ignored and will always be 'true' if used with the --
self-managed flag. (default true)
      --with-aws-bootstrap-credentials        Set true to use AWS bootstrap
credentials from your environment. When false, the instance profile of the EC2
instance where the CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials        Set true to use GCP bootstrap
credentials from your environment. When false, the service account of the VM instance
where the CAPG controller is scheduled on will be used instead.
      --worker-cpus int                       The number of virtual processors in a
worker machine (default 8)
      --worker-disk-size int                  The size of a worker machine's disk, in
GB (default 80)
      --worker-http-proxy string              HTTP proxy for nodes
      --worker-https-proxy string             HTTPS proxy for nodes
      --worker-memory int                     The size of a worker machine's memory,
in GB (default 32)
      --worker-no-proxy strings               No Proxy list for nodes (default [])
      --worker-replicas int                   Number of workers (default 4)
```

12.2.7.10.6.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.10.6.3 SEE ALSO

- dkp create cluster (see page 1853) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.10.7 dkp create cluster aks

Create a Konvoy cluster in AKS

```
dkp create cluster aks [flags]
```

12.2.7.10.7.1 Options

```
      --additional-tags stringToString     Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys        If true, ignore any errors in templates
when a field or map key is missing in the template. Only applies to golang and
jsonpath output formats. (default true)
      --aws-service-endpoints string       Custom AWS service endpoints in a semi-
colon separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=${URL};$
{SigningRegion2}...
      --certificate-renew-interval int     The interval number of days Kubernetes
managed PKI certificates are renewed. For example, an Interval value of 30 means the
certificates will be refreshed every 30 days. A value of 0 disables the feature. (def
ault 0)
  -c, --cluster-name name                  Name used to prefix the cluster and all
the created resources.
      --dry-run                            Only print the objects that would be
created, without creating them.
  -h, --help                               help for aks
      --http-proxy string                  HTTP proxy for CAPI controllers
      --https-proxy string                 HTTPS proxy for CAPI controllers
      --kind-cluster-image string          Kind node image for the bootstrap cluster
(default "mesosphere/konvoy-bootstrap:v2.8.1")
      --kubeconfig string                  Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-pod-network-cidr cidr   The Kubernetes Pod network CIDR to use in
the cluster (default 192.168.0.0/16)
      --kubernetes-service-cidr cidr       The Kubernetes Service CIDR to use in the
cluster (default 10.96.0.0/12)
      --kubernetes-version string          Kubernetes version. Run 'az aks get-
versions -o table --location <location>' to see available versions. See https://
docs.microsoft.com/en-us/azure/aks/supported-kubernetes-versions for more details.
Must be a patch version for v1.28.x.
      --location string                    Azure location to deploy cluster to
(default "westus")
  -n, --namespace string                   If present, the namespace scope for this
 CLI request. (default "default")
      --no-proxy strings                   No Proxy list for CAPI controllers (defaul
t [])
  -o, --output string                      Output format. One of: (json, yaml, name,
go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string            Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --show-managed-fields                If true, keep the managedFields when
printing objects in JSON or YAML format.
```

```
      --ssh-public-key-file string         Path to the authorized SSH key for the
user
      --ssh-username string                Name of the user to create on the instance
(default "konvoy")
      --system-machine-size string         System node pool machine size (default
"Standard_D4s_v3")
      --system-replicas int                Number of system nodes (default 3)
      --template string                    Template string or path to template file
to use when -o=go-template, -o=go-template-file. The template format is golang
templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                   The length of time to wait before giving
up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                               If true, wait for operations to complete
before returning. (default true)
      --with-aws-bootstrap-credentials     Set true to use AWS bootstrap credentials
from your environment. When false, the instance profile of the EC2 instance where the
CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials     Set true to use GCP bootstrap credentials
from your environment. When false, the service account of the VM instance where the
CAPG controller is scheduled on will be used instead.
      --worker-availability-zone string    The availability zone in the region to
deploy the worker nodes to, if not set a random one will be selected (ex. 1). Not all
locations, including the default 'westus', support setting this flag, see https://
docs.microsoft.com/en-us/azure/availability-zones/az-overview.
      --worker-http-proxy string           HTTP proxy for nodes
      --worker-https-proxy string          HTTPS proxy for nodes
      --worker-machine-size string         Worker machine size (default
"Standard_D8s_v3")
      --worker-no-proxy strings            No Proxy list for nodes (default [])
      --worker-replicas int                Number of workers (default 4)
```

12.2.7.10.7.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.10.7.3 SEE ALSO

- dkp create cluster (see page 1853) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.10.8 dkp create cluster preprovisioned

Create a Konvoy cluster on pre-provisioned infrastructure

```
dkp create cluster preprovisioned [flags]
```

12.2.7.10.8.1 Options

```
      --allow-missing-template-keys             If true, ignore any errors in
templates when a field or map key is missing in the template. Only applies to golang
and jsonpath output formats. (default true)
      --aws-service-endpoints string            Custom AWS service endpoints in a
semi-colon separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=$
{URL};${SigningRegion2}...
      --certificate-renew-interval int          The interval number of days
Kubernetes managed PKI certificates are renewed. For example, an Interval value of 30
 means the certificates will be refreshed every 30 days. A value of 0 disables the
feature. (default 0)
  -c, --cluster-name name                       Name used to prefix the cluster and
all the created resources.
      --control-plane-endpoint-host string      The control plane endpoint address.
To use an external load balancer, set to its IP or hostname. To use the built-in
virtual IP, set to a static IPv4 address in the Layer 2 network of the control plane
machines. [Not for production use: To use a single-machine control plane, set to the
IP or hostname of the machine.]
      --control-plane-endpoint-port int         The control plane endpoint port. To
use an external load balancer, set to its listening port. (default 6443)
      --control-plane-http-proxy string         HTTP proxy for control plane machines
      --control-plane-https-proxy string        HTTPS proxy for control plane
machines
      --control-plane-no-proxy strings          No Proxy list for control plane
machines (default [])
      --control-plane-replicas int              Number of control plane nodes (defaul
t 3)
      --dry-run                                 Only print the objects that would be
created, without creating them.
      --etcd-image-repository string            The image repository to use for
 pulling the etcd image
      --etcd-version string                     The version of etcd to use.
      --extra-sans strings                      A comma separated list of additional
Subject Alternative Names for the API Server signing cert (default [])
  -h, --help                                    help for preprovisioned
      --http-proxy string                       HTTP proxy for CAPI controllers
      --https-proxy string                      HTTPS proxy for CAPI controllers
      --kind-cluster-image string               Kind node image for the bootstrap
cluster (default "mesosphere/konvoy-bootstrap:v2.8.1")
      --kubeconfig string                       Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply. This flag is
ignored if used with the --self-managed flag.
      --kubernetes-image-repository string      The image repository to use for
 pulling kubernetes images
      --kubernetes-pod-network-cidr cidr        The Kubernetes Pod network CIDR to
use in the cluster (default 192.168.0.0/16)
      --kubernetes-service-cidr cidr            The Kubernetes Service CIDR to use in
the cluster (default 10.96.0.0/12)
```

```
      --kubernetes-version string               Kubernetes version (default "1.28.7")
  -n, --namespace string                        If present, the namespace scope for
this CLI request. (default "default")
      --no-proxy strings                        No Proxy list for CAPI controllers (d
efault [])
  -o, --output string                           Output format. One of: (json, yaml,
name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-
json, jsonpath-file).
      --output-directory string                 Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --override-secret-name string             Name of the secret for any provided
overrides on a preprovisioned cluster. All overrides defined at provisioning should
be present in this secret.
      --pre-provisioned-inventory-file string   Path to PreprovisionedInventory
inventory file
      --registry-mirror-cacert file             CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string         Password to authenticate to the
registry mirror with
      --registry-mirror-url url                 URL of a container registry to use as
a mirror in the cluster
      --registry-mirror-username string         Username to authenticate to the
registry mirror with
      --self-managed                            When set to true, the required
prerequisites are created before creating the cluster and the resulting cluster has
all necessary components deployed onto itself, so it can manage its own cluster
lifecycle. When set to false, a management cluster is used. (default false)
      --show-managed-fields                     If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-private-key-file string             Path to the private SSH key file used
by Konvoy to access the pre-provisioned hosts
      --ssh-public-key-file string              Path to the authorized SSH key for
 the user
      --ssh-username string                     Name of the user to create on the
instance (default "konvoy")
      --template string                         Template string or path to template
file to use when -o=go-template, -o=go-template-file. The template format is golang
templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                        The length of time to wait before
giving up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --virtual-ip-interface string             The network interface, e.g, 'eth0' or
'ens5', to use for the built-in virtual IP control plane endpoint. This interface
 must be available on every control plane machine. If the value is empty, the flag
does nothing. If the value is not empty, the built-in virtual IP control plane
endpoint is created, using values from --control-plane-endpoint-host and --control-
plane-endpoint-port.
      --wait                                    If true, wait for operations to
complete before returning. This flag is ignored and will always be 'true' if used
with the --self-managed flag. (default true)
      --with-aws-bootstrap-credentials          Set true to use AWS bootstrap
credentials from your environment. When false, the instance profile of the EC2
instance where the CAPA controller is scheduled on will be used instead.
```

```
      --with-gcp-bootstrap-credentials          Set true to use GCP bootstrap
credentials from your environment. When false, the service account of the VM instance
where the CAPG controller is scheduled on will be used instead.
      --worker-http-proxy string                HTTP proxy for nodes
      --worker-https-proxy string               HTTPS proxy for nodes
      --worker-no-proxy strings                 No Proxy list for nodes (default [])
      --worker-replicas int                     Number of workers (default 4)
```

12.2.7.10.8.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.10.8.3 SEE ALSO

- dkp create cluster (see page 1853) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.10.9 dkp create cluster gcp

Create a Konvoy cluster in GCP

```
dkp create cluster gcp [flags]
```

12.2.7.10.9.1 Options

```
      --additional-tags stringToString              Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys                 If true, ignore any errors in
templates when a field or map key is missing in the template. Only applies to golang
and jsonpath output formats. (default true)
      --associate-public-ip-address                 Associate a public IP for all
machines. When set to false the specified network must have Cloud NAT configured to
provide internet access. (default true)
      --aws-service-endpoints string                Custom AWS service endpoints in
a semi-colon separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=$
{URL};${SigningRegion2}...
      --certificate-renew-interval int              The interval number of days
Kubernetes managed PKI certificates are renewed. For example, an Interval value of 30
 means the certificates will be refreshed every 30 days. A value of 0 disables the
feature. (default 0)
  -c, --cluster-name name                           Name used to prefix the cluster
and all the created resources.
```

```
      --control-plane-http-proxy string              HTTP proxy for control plane
machines
      --control-plane-https-proxy string             HTTPS proxy for control plane
machines
      --control-plane-instance-type string           Control Plane machine instance
type (default "n2-standard-4")
      --control-plane-no-proxy strings               No Proxy list for control plane
machines (default [])
      --control-plane-replicas int                   Number of control plane nodes (d
efault 3)
      --control-plane-service-account-email string   Control Plane Service Account
email address (default "default")
      --dry-run                                      Only print the objects that
would be created, without creating them.
      --etcd-image-repository string                 The image repository to use for
 pulling the etcd image
      --etcd-version string                          The version of etcd to use.
      --extra-sans strings                           A comma separated list of
additional Subject Alternative Names for the API Server signing cert (default [])
  -h, --help                                         help for gcp
      --http-proxy string                            HTTP proxy for CAPI controllers
      --https-proxy string                           HTTPS proxy for CAPI controllers
      --image string                                 Full reference to an image to
use for all nodes (set either this or --image-family) (ex. 'projects/my-project/
global/images/konvoy-ubuntu-2004-1-99-99-1234567890')
      --image-family string                          Full reference to an image
family to use for all nodes (set either this or --image) (ex. 'projects/my-project/
global/images/family/konvoy-ubuntu-2004-{{.K8sVersion}}')
      --kind-cluster-image string                    Kind node image for the
bootstrap cluster (default "mesosphere/konvoy-bootstrap:v2.8.1")
      --kubeconfig string                            Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply. This flag is
ignored if used with the --self-managed flag.
      --kubernetes-image-repository string           The image repository to use for
 pulling kubernetes images
      --kubernetes-pod-network-cidr cidr             The Kubernetes Pod network CIDR
to use in the cluster (default 192.168.0.0/16)
      --kubernetes-service-cidr cidr                 The Kubernetes Service CIDR to
use in the cluster (default 10.96.0.0/12)
      --kubernetes-version string                    Kubernetes version (default
"1.28.7")
  -n, --namespace string                             If present, the namespace scope
for this CLI request. (default "default")
      --network string                               The GCP network name to deploy
the cluster to (default "default")
      --no-proxy strings                             No Proxy list for CAPI
controllers (default [])
  -o, --output string                                Output format. One of: (json,
yaml, name, go-template, go-template-file, template, templatefile, jsonpath,
jsonpath-as-json, jsonpath-file).
      --output-directory string                      Used with --output=json|yaml.
The directory where to output resources to files. The directory must already exist.
```

```
      --project string                                The GCP project name to deploy
the cluster to
      --region string                                 GCP region to deploy cluster to
(default "us-west1")
      --registry-mirror-cacert file                   CA certificate chain to use
while communicating with the registry mirror using TLS
      --registry-mirror-password string               Password to authenticate to the
registry mirror with
      --registry-mirror-url url                       URL of a container registry to
use as a mirror in the cluster
      --registry-mirror-username string               Username to authenticate to the
registry mirror with
      --self-managed                                  When set to true, the required
prerequisites are created before creating the cluster and the resulting cluster has
all necessary components deployed onto itself, so it can manage its own cluster
lifecycle. When set to false, a management cluster is used. (default false)
      --show-managed-fields                           If true, keep the managedFields
when printing objects in JSON or YAML format.
      --ssh-public-key-file string                    Path to the authorized SSH key
for the user
      --ssh-username string                           Name of the user to create on
the instance (default "konvoy")
      --template string                               Template string or path to
template file to use when -o=go-template, -o=go-template-file. The template format is
golang templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                              The length of time to wait
before giving up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                                          If true, wait for operations to
complete before returning. This flag is ignored and will always be 'true' if used
with the --self-managed flag. (default true)
      --with-aws-bootstrap-credentials                Set true to use AWS bootstrap
credentials from your environment. When false, the instance profile of the EC2
instance where the CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials                Set true to use GCP bootstrap
credentials from your environment. When false, the service account of the VM instance
where the CAPG controller is scheduled on will be used instead.
      --worker-http-proxy string                      HTTP proxy for nodes
      --worker-https-proxy string                     HTTPS proxy for nodes
      --worker-instance-type string                   Worker machine instance type
(default "n2-standard-8")
      --worker-no-proxy strings                       No Proxy list for nodes (default
 [])
      --worker-replicas int                           Number of workers (default 4)
      --worker-service-account-email string           Worker machine Service Account
email address (default "default")
      --worker-zone string                            Zone in the region to deploy the
worker nodes to, if not set a random one will be selected (ex. us-west1-a)
```

12.2.7.10.9.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.10.9.3 SEE ALSO

- dkp create cluster (see page 1853) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.10.10 dkp create cluster vcd

Create a Konvoy cluster in VMware Cloud Director

```
dkp create cluster vcd [flags]
```

12.2.7.10.10.1 Options

```
      --allow-missing-template-keys             If true, ignore any errors in
templates when a field or map key is missing in the template. Only applies to golang
and jsonpath output formats. (default true)
      --aws-service-endpoints string            Custom AWS service endpoints in a
semi-colon separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=$
{URL};${SigningRegion2}...
      --catalog string                          Catalog hosting the virtual machine
vApp template.
      --certificate-renew-interval int          The interval number of days
Kubernetes managed PKI certificates are renewed. For example, an Interval value of 30
 means the certificates will be refreshed every 30 days. A value of 0 disables the
feature. (default 0)
  -c, --cluster-name name                       Name used to prefix the cluster and
all the created resources.
      --control-plane-endpoint-host string      The control plane endpoint IP or
hostname.
      --control-plane-endpoint-port int         The control plane endpoint port. To
use an external load balancer, set to its listening port. (default 6443)
      --control-plane-http-proxy string         HTTP proxy for control plane machines
      --control-plane-https-proxy string        HTTPS proxy for control plane
machines
      --control-plane-no-proxy strings          No Proxy list for control plane
machines (default [])
      --control-plane-placement-policy string   The placement policy for control
plane to be used on this machine
```

```
      --control-plane-replicas int                Number of control plane nodes (defaul
t 3)
      --control-plane-sizing-policy string        The sizing policy for control plane
to be used on this machine
      --data-center string                        The organization's virtual datacenter
name where the cluster should be deployed.
      --dry-run                                   Only print the objects that would be
created, without creating them.
      --etcd-image-repository string              The image repository to use for
 pulling the etcd image
      --etcd-version string                       The version of etcd to use.
      --extra-sans strings                        A comma separated list of additional
Subject Alternative Names for the API Server signing cert (default [])
  -h, --help                                      help for vcd
      --http-proxy string                         HTTP proxy for CAPI controllers
      --https-proxy string                        HTTPS proxy for CAPI controllers
      --kind-cluster-image string                 Kind node image for the bootstrap
cluster (default "mesosphere/konvoy-bootstrap:v2.8.1")
      --kubeconfig string                         Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply. This flag is
ignored if used with the --self-managed flag.
      --kubernetes-image-repository string        The image repository to use for
 pulling kubernetes images
      --kubernetes-pod-network-cidr cidr          The Kubernetes Pod network CIDR to
use in the cluster (default 192.168.0.0/16)
      --kubernetes-service-cidr cidr              The Kubernetes Service CIDR to use in
the cluster (default 10.96.0.0/12)
      --kubernetes-version string                 Kubernetes version (default "1.28.7")
      --loadbalancer-network-cidr cidr            The load balancer CIDR to use for
 both the control plane nodes and kubernetes external services
  -n, --namespace string                          If present, the namespace scope for
this CLI request. (default "default")
      --network string                            The organization's virtual datacenter
network to be used by the cluster.
      --no-proxy strings                          No Proxy list for CAPI controllers (d
efault [])
      --organization string                       The organization name where the
cluster should be deployed.
  -o, --output string                             Output format. One of: (json, yaml,
name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-
json, jsonpath-file).
      --output-directory string                   Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --registry-mirror-cacert file               CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string           Password to authenticate to the
registry mirror with
      --registry-mirror-url url                   URL of a container registry to use as
a mirror in the cluster
      --registry-mirror-username string           Username to authenticate to the
registry mirror with
```

```
      --self-managed                              When set to true, the required
prerequisites are created before creating the cluster and the resulting cluster has
all necessary components deployed onto itself, so it can manage its own cluster
lifecycle. When set to false, a management cluster is used. (default false)
      --show-managed-fields                       If true, keep the managedFields when
printing objects in JSON or YAML format.
      --site url                                  The cloud director's endpoint with
the format https://VCD_HOST.
      --ssh-public-key-file string                Path to the authorized SSH key for
 the user
      --ssh-username string                       Name of the user to create on the
instance (default "konvoy")
      --storage-profile string                    The storage profile to be used for a
virtual machine (default "*")
      --template string                           Template string or path to template
file to use when -o=go-template, -o=go-template-file. The template format is golang
templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                          The length of time to wait before
giving up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --vapp-template string                      The vApp template to use for a
virtual machine.
      --wait                                      If true, wait for operations to
complete before returning. This flag is ignored and will always be 'true' if used
with the --self-managed flag. (default true)
      --with-aws-bootstrap-credentials            Set true to use AWS bootstrap
credentials from your environment. When false, the instance profile of the EC2
instance where the CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials            Set true to use GCP bootstrap
credentials from your environment. When false, the service account of the VM instance
where the CAPG controller is scheduled on will be used instead.
      --worker-http-proxy string                  HTTP proxy for nodes
      --worker-https-proxy string                 HTTPS proxy for nodes
      --worker-no-proxy strings                   No Proxy list for nodes (default [])
      --worker-placement-policy string            The placement policy for workers to
be used on this machine
      --worker-replicas int                       Number of workers (default 4)
      --worker-sizing-policy string               The sizing policy for workers to be
used on this machine
```

12.2.7.10.10.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.10.10.3 SEE ALSO

- dkp create cluster (see page 1853) - Create a Kubernetes cluster, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.11 dkp create nodepool

Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.11.1 Options

```
  -h, --help          help for nodepool
  -v, --verbose int   Output verbosity
```

12.2.7.11.2 SEE ALSO

- dkp create (see page 1848) - Create one of [appdeployment, bootstrap, capi-components, chart-bundle, cluster, image-bundle, nodepool, workspace]
- dkp create nodepool aks (see page 1879) - Create a nodepool in AKS
- dkp create nodepool aws (see page 1880) - Create a nodepool in AWS
- dkp create nodepool azure (see page 1875) - Create a nodepool in Azure
- dkp create nodepool eks (see page 1882) - Create a nodepool for EKS
- dkp create nodepool gcp (see page 1883) - Create a nodepool in GCP
- dkp create nodepool preprovisioned (see page 1885) - Create a nodepool for a Pre Provisioned Cluster.
- dkp create nodepool vcd (see page 1887) - Create a nodepool in VCD
- dkp create nodepool vsphere (see page 1877) - Create a nodepool in vSphere

12.2.7.11.3 dkp create nodepool azure

Create a nodepool in Azure

```
dkp create nodepool azure name [flags]
```

12.2.7.11.3.1 Options

```
      --additional-tags stringToString    Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys       If true, ignore any errors in templates
when a field or map key is missing in the template. Only applies to golang and
jsonpath output formats. (default true)
      --availability-zone string          The availability zone in the region to
deploy the worker nodes to, if not set a random one will be selected (ex. 1). Not all
locations, including the default 'westus', support setting this flag, see https://
docs.microsoft.com/en-us/azure/availability-zones/az-overview.
```

```
  -c, --cluster-name name                 Name used to prefix the cluster and all the
created resources.
      --compute-gallery-id string         Compute Gallery ID of a custom image, e.g.,
'/subscriptions/<subscription id>/resourceGroups/<resource group name>/providers/
Microsoft.Compute/galleries/<gallery name>/images/<image definition name>/versions/
<version id>' (replacing placeholders with the values used when creating the image)
      --dry-run                           Only print the objects that would be
created, without creating them.
  -h, --help                              help for azure
      --http-proxy string                 HTTP proxy for nodes
      --https-proxy string                HTTPS proxy for nodes
      --kubeconfig string                 Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string         Kubernetes version (default "1.28.7")
      --machine-size string               Worker machine size (default
"Standard_D8s_v3")
  -n, --namespace string                  If present, the namespace scope for this
 CLI request. (default "default")
      --no-proxy strings                  No Proxy list for nodes (default [])
  -o, --output string                     Output format. One of: (json, yaml, name,
go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string           Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --plan-offer string                 The offer for a Marketplace image or a
custom image sourced from a Marketplace image requiring Plan information.
      --plan-publisher string             The publisher for a Marketplace image or a
custom image sourced from a Marketplace image requiring Plan information.
      --plan-sku string                   The SKU for a Marketplace image or a custom
image sourced from a Marketplace image requiring Plan information.
      --registry-mirror-cacert file       CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string   Password to authenticate to the registry
mirror with
      --registry-mirror-url url           URL of a container registry to use as a
mirror in the cluster
      --registry-mirror-username string   Username to authenticate to the registry
mirror with
      --replicas int                      Number of replicas (default 1)
      --show-managed-fields               If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string        Path to the authorized SSH key for the user
      --ssh-username string               Name of the user to create on the instance
(default "konvoy")
      --template string                   Template string or path to template file to
use when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                  The length of time to wait before giving
up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                              If true, wait for operations to complete
before returning.
```

12.2.7.11.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.11.3.3 SEE ALSO

- dkp create nodepool (see page 1875) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.11.4 dkp create nodepool vsphere

Create a nodepool in vSphere

```
dkp create nodepool vsphere name [flags]
```

12.2.7.11.4.1 Options

```
      --additional-tags stringToString    Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys       If true, ignore any errors in templates
when a field or map key is missing in the template. Only applies to golang and
jsonpath output formats. (default true)
  -c, --cluster-name name                 Name used to prefix the cluster and all the
created resources.
      --cpus int                          The number of virtual processors in a
worker machine (default 8)
      --data-center string                The vSphere datacenter to deploy the
workload cluster on.
      --data-store string                 The vSphere datastore to deploy the
workload cluster on.
      --disk-size int                     The size of a worker machine's disk, in GB
(default 80)
      --dry-run                           Only print the objects that would be
created, without creating them.
      --folder string                     The vSphere folder for your VMs. Set to ""
 to use the root vSphere folder.
  -h, --help                              help for vsphere
      --http-proxy string                 HTTP proxy for nodes
      --https-proxy string                HTTPS proxy for nodes
      --kubeconfig string                 Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string         Kubernetes version (default "1.28.7")
```

```
      --memory int                          The size of a worker machine's memory, in
GB (default 32)
  -n, --namespace string                    If present, the namespace scope for this
 CLI request. (default "default")
      --network string                      The vSphere network to deploy the workload
cluster on.
      --no-proxy strings                    No Proxy list for nodes (default [])
      --os-hint flatcar                     A hint which will allow the installer to
generate appropriate configurations for a target OS. Presently, only the hint for
 flatcar is supported.
  -o, --output string                       Output format. One of: (json, yaml, name,
go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string             Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --registry-mirror-cacert file         CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string     Password to authenticate to the registry
mirror with
      --registry-mirror-url url             URL of a container registry to use as a
mirror in the cluster
      --registry-mirror-username string     Username to authenticate to the registry
mirror with
      --replicas int                        Number of replicas (default 1)
      --resource-pool string                The vSphere resource pool for the workload
cluster's virtual machines.
      --server string                       The vCenter server address. Accepted
formats: host, host:port, http[s]://host[:port]. Accepted host formats: IPv4, IPv6,
or DNS name.
      --show-managed-fields                 If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string          Path to the authorized SSH key for the user
      --ssh-username string                 Name of the user to create on the instance
(default "konvoy")
      --storage-policy string               This is the vSphere storage policy. Set it
to "" if you don't want to use a storage policy.
      --template string                     Template string or path to template file to
use when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                    The length of time to wait before giving
up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --tls-thumb-print string              sha1 thumbprint of the vcenter certificate:
openssl x509 -sha1 -fingerprint -in ca.crt -noout
      --vm-template string                  The virtual machine template to use for the
workload cluster's virtual machines.
      --wait                                If true, wait for operations to complete
before returning.
```

12.2.7.11.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.11.4.3 SEE ALSO

- dkp create nodepool (see page 1875) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.11.5 dkp create nodepool aks

Create a nodepool in AKS

```
dkp create nodepool aks name [flags]
```

12.2.7.11.5.1 Options

```
      --additional-tags stringToString   Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys      If true, ignore any errors in templates when
a field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
      --availability-zone string         The availability zone in the region to
deploy the worker nodes to, if not set a random one will be selected (ex. 1). Not all
locations, including the default 'westus', support setting this flag, see https://
docs.microsoft.com/en-us/azure/availability-zones/az-overview.
  -c, --cluster-name name                Name used to prefix the cluster and all the
created resources.
      --dry-run                          Only print the objects that would be
created, without creating them.
  -h, --help                             help for aks
      --http-proxy string                HTTP proxy for nodes
      --https-proxy string               HTTPS proxy for nodes
      --kubeconfig string                Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string        Kubernetes version. Run 'az aks get-versions
-o table --location <location>' to see available versions. See https://
docs.microsoft.com/en-us/azure/aks/supported-kubernetes-versions for more details.
Must be a patch version for v1.28.x.
      --machine-size string              Worker machine size (default
"Standard_D8s_v3")
  -n, --namespace string                 If present, the namespace scope for this CLI
request. (default "default")
```

```
      --no-proxy strings                No Proxy list for nodes (default [])
  -o, --output string                   Output format. One of: (json, yaml, name,
go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string         Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --replicas int                    Number of replicas (default 1)
      --show-managed-fields             If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string      Path to the authorized SSH key for the user
      --ssh-username string             Name of the user to create on the instance
(default "konvoy")
      --template string                 Template string or path to template file to
use when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                The length of time to wait before giving up.
Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                            If true, wait for operations to complete
before returning.
```

12.2.7.11.5.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.11.5.3 SEE ALSO

- dkp create nodepool (see page 1875) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.11.6 dkp create nodepool aws

Create a nodepool in AWS

```
dkp create nodepool aws [flags]
```

12.2.7.11.6.1 Options

```
      --additional-security-group-ids strings   A comma separated list of existing
security group IDs to use for machines in addition to those created automatically (de
fault [])
      --additional-tags stringToString          Tags to apply to the provisioned
infrastructure (default [])
```

```
      --allow-missing-template-keys             If true, ignore any errors in
templates when a field or map key is missing in the template. Only applies to golang
and jsonpath output formats. (default true)
      --ami string                              AMI ID to use for machines
      --ami-base-os string                      Base OS used in search of AMIs.
Examples: 'centos-7', 'ubuntu-18.04', 'ubuntu-20.04'
      --ami-format string                       Query string used in search of AMIs.
Example: When --ami-base-os='rhel8.4', then the string 'prefix-{{.BaseOS}}-?
{{.K8sVersion}}-*' matches any AMIs with the Name 'prefix-rhel8.4-1.28.7
      --ami-owner string                        ID of AWS account used in search of
AMIs
      --availability-zone string                The AvailabilityZone in the region to
deploy the worker nodes to, if not set a random one will be selected (ex. us-west-2a)
  -c, --cluster-name name                       Name used to prefix the cluster and
all the created resources.
      --dry-run                                 Only print the objects that would be
created, without creating them.
  -h, --help                                    help for aws
      --http-proxy string                       HTTP proxy for nodes
      --https-proxy string                      HTTPS proxy for nodes
      --iam-instance-profile string             Name of the IAM instance profile to
assign to worker machines. (default "nodes.cluster-api-provider-aws.sigs.k8s.io")
      --instance-type string                    Worker machine instance type (default
 "m5.2xlarge")
      --kubeconfig string                       Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string               Kubernetes version (default "1.28.7")
  -n, --namespace string                        If present, the namespace scope for
this CLI request. (default "default")
      --no-proxy strings                        No Proxy list for nodes (default [])
      --os-hint flatcar                         A hint which will allow the installer
to generate appropriate configurations for a target OS. Presently, only the hint for
 flatcar is supported.
  -o, --output string                           Output format. One of: (json, yaml,
name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-
json, jsonpath-file).
      --output-directory string                 Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --registry-mirror-cacert file             CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string         Password to authenticate to the
registry mirror with
      --registry-mirror-url url                 URL of a container registry to use as
a mirror in the cluster
      --registry-mirror-username string         Username to authenticate to the
registry mirror with
      --replicas int                            Number of replicas (default 1)
      --show-managed-fields                     If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string              Path to the authorized SSH key for
 the user
```

```
      --ssh-username string                    Name of the user to create on the
instance (default "konvoy")
      --template string                        Template string or path to template
file to use when -o=go-template, -o=go-template-file. The template format is golang
templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                       The length of time to wait before
giving up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                                   If true, wait for operations to
complete before returning.
```

12.2.7.11.6.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.11.6.3 SEE ALSO

- dkp create nodepool (see page 1875) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.11.7 dkp create nodepool eks

Create a nodepool for EKS

```
dkp create nodepool eks [flags]
```

12.2.7.11.7.1 Options

```
      --additional-security-group-ids strings   A comma separated list of existing
security group IDs to use for machines in addition to those created automatically (de
fault [])
      --additional-tags stringToString          Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys             If true, ignore any errors in
templates when a field or map key is missing in the template. Only applies to golang
and jsonpath output formats. (default true)
      --availability-zone string                The AvailabilityZone in the region to
deploy the worker nodes to, if not set a random one will be selected (ex. us-west-2a)
  -c, --cluster-name name                       Name used to prefix the cluster and
all the created resources.
      --dry-run                                 Only print the objects that would be
created, without creating them.
  -h, --help                                    help for eks
      --http-proxy string                       HTTP proxy for nodes
```

```
      --https-proxy string                        HTTPS proxy for nodes
      --iam-instance-profile string               Name of the IAM instance profile to
assign to worker machines. (default "nodes.cluster-api-provider-aws.sigs.k8s.io")
      --instance-type string                      Worker machine instance type (default
 "m5.2xlarge")
      --kubeconfig string                         Path to the kubeconfig for the
management cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string                 Kubernetes version (default "1.27.9")
  -n, --namespace string                          If present, the namespace scope for
this CLI request. (default "default")
      --no-proxy strings                          No Proxy list for nodes (default [])
  -o, --output string                             Output format. One of: (json, yaml,
name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-
json, jsonpath-file).
      --output-directory string                   Used with --output=json|yaml. The
directory where to output resources to files. The directory must already exist.
      --replicas int                              Number of replicas (default 1)
      --show-managed-fields                       If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string                Path to the authorized SSH key for
 the user
      --ssh-username string                       Name of the user to create on the
instance (default "konvoy")
      --template string                           Template string or path to template
file to use when -o=go-template, -o=go-template-file. The template format is golang
templates [http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                          The length of time to wait before
giving up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                                      If true, wait for operations to
complete before returning.
```

12.2.7.11.7.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.11.7.3 SEE ALSO

- dkp create nodepool (see page 1875) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.11.8 dkp create nodepool gcp

Create a nodepool in GCP

```
dkp create nodepool gcp name [flags]
```

12.2.7.11.8.1 Options

```
      --additional-tags stringToString    Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys       If true, ignore any errors in templates
when a field or map key is missing in the template. Only applies to golang and
jsonpath output formats. (default true)
      --associate-public-ip-address       Associate a public IP for all machines.
When set to false the specified network must have Cloud NAT configured to provide
internet access. (default true)
  -c, --cluster-name name                 Name used to prefix the cluster and all the
created resources.
      --dry-run                           Only print the objects that would be
created, without creating them.
  -h, --help                              help for gcp
      --http-proxy string                 HTTP proxy for nodes
      --https-proxy string                HTTPS proxy for nodes
      --image string                      Full reference to an image to use for all
nodes (set either this or --image-family) (ex. 'projects/my-project/global/images/
konvoy-ubuntu-2004-1-99-99-1234567890')
      --image-family string               Full reference to an image family to use
for all nodes (set either this or --image) (ex. 'projects/my-project/global/images/
family/konvoy-ubuntu-2004-{{.K8sVersion}}')
      --instance-type string              Worker machine instance type (default "n2-
standard-8")
      --kubeconfig string                 Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string         Kubernetes version (default "1.28.7")
  -n, --namespace string                  If present, the namespace scope for this
 CLI request. (default "default")
      --no-proxy strings                  No Proxy list for nodes (default [])
  -o, --output string                     Output format. One of: (json, yaml, name,
go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string           Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --registry-mirror-cacert file       CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string   Password to authenticate to the registry
mirror with
      --registry-mirror-url url           URL of a container registry to use as a
mirror in the cluster
      --registry-mirror-username string   Username to authenticate to the registry
mirror with
      --replicas int                      Number of replicas (default 1)
      --service-account-email string      Worker machine Service Account email
address (default "default")
      --show-managed-fields               If true, keep the managedFields when
printing objects in JSON or YAML format.
```

```
      --ssh-public-key-file string        Path to the authorized SSH key for the user
      --ssh-username string               Name of the user to create on the instance
(default "konvoy")
      --template string                   Template string or path to template file to
use when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                  The length of time to wait before giving
up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                              If true, wait for operations to complete
before returning.
      --zone string                       Zone in the region to deploy the worker
nodes to, if not set a random one will be selected (ex. us-west1-a)
```

12.2.7.11.8.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.11.8.3 SEE ALSO

- dkp create nodepool (see page 1875) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.11.9 dkp create nodepool preprovisioned

Create a nodepool for a Pre Provisioned Cluster.

```
dkp create nodepool preprovisioned name [flags]
```

12.2.7.11.9.1 Options

```
      --additional-tags stringToString    Tags to apply to the provisioned
infrastructure (default [])
      --allow-missing-template-keys       If true, ignore any errors in templates
when a field or map key is missing in the template. Only applies to golang and
jsonpath output formats. (default true)
  -c, --cluster-name name                 Name used to prefix the cluster and all the
created resources.
      --dry-run                           Only print the objects that would be
created, without creating them.
  -h, --help                              help for preprovisioned
      --http-proxy string                 HTTP proxy for nodes
      --https-proxy string                HTTPS proxy for nodes
```

```
      --kubeconfig string                  Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string          Kubernetes version (default "1.28.7")
  -n, --namespace string                   If present, the namespace scope for this
 CLI request. (default "default")
      --no-proxy strings                   No Proxy list for nodes (default [])
  -o, --output string                      Output format. One of: (json, yaml, name,
go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string            Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --override-secret-name string        Name of the secret for any provided
overrides on a preprovisioned cluster. All overrides defined at provisioning should
be present in this secret.
      --registry-mirror-cacert file        CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string    Password to authenticate to the registry
mirror with
      --registry-mirror-url url            URL of a container registry to use as a
mirror in the cluster
      --registry-mirror-username string    Username to authenticate to the registry
mirror with
      --replicas int                       Number of replicas (default 1)
      --show-managed-fields                If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string         Path to the authorized SSH key for the user
      --ssh-username string                Name of the user to create on the instance
(default "konvoy")
      --template string                    Template string or path to template file to
use when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                   The length of time to wait before giving
up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --wait                               If true, wait for operations to complete
before returning.
```

12.2.7.11.9.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.11.9.3 SEE ALSO

- dkp create nodepool (see page 1875) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.7.11.10 dkp create nodepool vcd

Create a nodepool in VCD

```
dkp create nodepool vcd name [flags]
```

12.2.7.11.10.1 Options

```
      --allow-missing-template-keys       If true, ignore any errors in templates
when a field or map key is missing in the template. Only applies to golang and
jsonpath output formats. (default true)
      --catalog string                    Catalog hosting the virtual machine vApp
template.
  -c, --cluster-name name                 Name used to prefix the cluster and all the
created resources.
      --dry-run                           Only print the objects that would be
created, without creating them.
  -h, --help                              help for vcd
      --http-proxy string                 HTTP proxy for nodes
      --https-proxy string                HTTPS proxy for nodes
      --kubeconfig string                 Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string         Kubernetes version (default "1.28.7")
  -n, --namespace string                  If present, the namespace scope for this
 CLI request. (default "default")
      --no-proxy strings                  No Proxy list for nodes (default [])
  -o, --output string                     Output format. One of: (json, yaml, name,
go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string           Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --registry-mirror-cacert file       CA certificate chain to use while
 communicating with the registry mirror using TLS
      --registry-mirror-password string   Password to authenticate to the registry
mirror with
      --registry-mirror-url url           URL of a container registry to use as a
mirror in the cluster
      --registry-mirror-username string   Username to authenticate to the registry
mirror with
      --replicas int                      Number of replicas (default 1)
      --show-managed-fields               If true, keep the managedFields when
printing objects in JSON or YAML format.
      --ssh-public-key-file string        Path to the authorized SSH key for the user
      --ssh-username string               Name of the user to create on the instance
(default "konvoy")
      --storage-profile string            The storage profile to be used for a
virtual machine (default "*")
```

```
      --template string                   Template string or path to template file to
use when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
      --timeout duration                  The length of time to wait before giving
up. Zero means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --vapp-template string              The vApp template to use for a virtual
machine.
      --wait                              If true, wait for operations to complete
before returning.
      --worker-placement-policy string    The placement policy for workers to be used
on this machine
      --worker-sizing-policy string       The sizing policy for workers to be used on
this machine
```

12.2.7.11.10.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.7.11.10.3 SEE ALSO

- dkp create nodepool (see page 1875) - Create a nodepool, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.8 dkp delete

Delete one of [bootstrap, capi-components, chart, cluster, nodepool]

12.2.8.1 Options

```
  -h, --help   help for delete
```

12.2.8.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.8.3 SEE ALSO

- dkp delete bootstrap (see page 1889) - Delete bootstrap cluster
- dkp delete capi-components (see page 1889) - Delete the CAPI components from the cluster
- dkp delete chart (see page 1891) - Delete a chart from the repository

- dkp delete cluster (see page 1890) - Delete a Kubernetes cluster
- dkp delete nodepool (see page 1891) - Delete a nodepool for a given cluster

12.2.8.4 dkp delete bootstrap

Delete bootstrap cluster

```
dkp delete bootstrap [flags]
```

12.2.8.4.1 Options

```
  -h, --help                       help for bootstrap
      --kind-cluster-name string   Kind cluster name for the bootstrap cluster (defau
lt "konvoy-capi-bootstrapper")
      --kubeconfig string          Path to the kubeconfig for the management cluster.
If unspecified, default discovery rules apply.
      --timeout duration           The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 5m0s)
  -v, --verbose int                Output verbosity
      --wait                       If true, wait for operations to complete before
returning. (default true)
```

12.2.8.4.2 SEE ALSO

- dkp delete (see page 1888) - Delete one of [bootstrap, capi-components, chart, cluster, nodepool]

12.2.8.5 dkp delete capi-components

Delete the CAPI components from the cluster

```
dkp delete capi-components [flags]
```

12.2.8.5.1 Options

```
  -h, --help                 help for capi-components
      --kubeconfig string    Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
      --timeout duration     The length of time to wait before giving up. Zero means
wait forever (e.g. 300s, 30m, 3h). (default 5m0s)
  -v, --verbose int          Output verbosity
```

```
      --wait                If true, wait for operations to complete before
returning. (default true)
```

12.2.8.5.2 SEE ALSO

- dkp delete (see page 1888) - Delete one of [bootstrap, capi-components, chart, cluster, nodepool]

12.2.8.6 dkp delete cluster

Delete a Kubernetes cluster

```
dkp delete cluster [flags]
```

12.2.8.6.1 Options

```
      --aws-service-endpoints string     Custom AWS service endpoints in a semi-colon
separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=${URL};$
{SigningRegion2}...
  -c, --cluster-name name                Name used to prefix the cluster and all the
created resources.
      --delete-kubernetes-resources      Delete Kubernetes resources on the cluster
before deleting that cluster (Services with type LoadBalancer) (default true)
  -h, --help                             help for cluster
      --http-proxy string                HTTP proxy for CAPI controllers
      --https-proxy string               HTTPS proxy for CAPI controllers
      --kind-cluster-image string        Kind node image for the bootstrap cluster (d
efault "mesosphere/konvoy-bootstrap:v2.8.1")
      --kubeconfig string                Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string                 If present, the namespace scope for this CLI
request. (default "default")
      --no-proxy strings                 No Proxy list for CAPI controllers (default
 [])
      --self-managed                     When set to true, the required prerequisites
and resources are moved from the self managed cluster before deleting. When set to
false, the resources are assumed installed in a management cluster. (default false)
      --timeout duration                 The length of time to wait before giving up.
Zero means wait forever (e.g. 300s, 30m, 3h). (default 15m0s)
  -v, --verbose int                      Output verbosity
      --wait                             If true, wait for operations to complete
before returning. This flag is ignored and will always be 'true' if used with the --
self-managed flag. (default true)
      --with-aws-bootstrap-credentials   Set true to use AWS bootstrap credentials
from your environment. When false, the instance profile of the EC2 instance where the
CAPA controller is scheduled on will be used instead.
```

```
      --with-gcp-bootstrap-credentials   Set true to use GCP bootstrap credentials
from your environment. When false, the service account of the VM instance where the
CAPG controller is scheduled on will be used instead.
```

12.2.8.6.2 SEE ALSO

- dkp delete (see page 1888) - Delete one of [bootstrap, capi-components, chart, cluster, nodepool]

12.2.8.7 dkp delete chart

Delete a chart from the repository

```
dkp delete chart [chartName] [chartVersion] [flags]
```

12.2.8.7.1 Options

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for chart
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.8.7.2 SEE ALSO

- dkp delete (see page 1888) - Delete one of [bootstrap, capi-components, chart, cluster, nodepool]

12.2.8.8 dkp delete nodepool

Delete a nodepool for a given cluster

```
dkp delete nodepool name [flags]
```

12.2.8.8.1 Options

```
  -c, --cluster-name name   Name used to prefix the cluster and all the created
resources.
      --dry-run             Only print the objects that would be created, without
creating them.
  -h, --help                help for nodepool
      --kubeconfig string   Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
  -n, --namespace string    If present, the namespace scope for this CLI request. (de
fault "default")
  -v, --verbose int         Output verbosity
```

12.2.8.8.2 SEE ALSO

- dkp delete (see page 1888) - Delete one of [bootstrap, capi-components, chart, cluster, nodepool]

12.2.9 dkp describe

Describe one of [cluster]

12.2.9.1 Options

```
  -h, --help          help for describe
  -v, --verbose int   Output verbosity
```

12.2.9.2 SEE ALSO

- dkp describe cluster (see page 1892) - Describe a Kubernetes cluster status

12.2.9.3 dkp describe cluster

Describe a Kubernetes cluster status

```
dkp describe cluster [flags]
```

12.2.9.3.1 Options

```
  -c, --cluster-name name   Name used to prefix the cluster and all the created
resources.
  -h, --help                help for cluster
      --kubeconfig string   Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
  -n, --namespace string    If present, the namespace scope for this CLI request. (de
fault "default")
```

12.2.9.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.9.3.3 SEE ALSO

- dkp describe (see page 1892) - Describe one of [cluster]

12.2.10 dkp detach

Detach one of [cluster]

12.2.10.1 Options

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for detach
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.10.2 SEE ALSO

- dkp detach cluster (see page 1894) - Detach a cluster

12.2.10.3 dkp detach cluster

Detach a cluster

```
dkp detach cluster CLUSTER_NAME [flags]
```

12.2.10.3.1 Options

```
  -h, --help               help for cluster
      --timeout duration   The length of time to wait before giving up on a detach,
defaults to wait forever (default 0s)
      --wait               If true, wait for resources to be gone before returning.
This waits for finalizers. (default true)
  -w, --workspace string   Name of the workspace of the attached cluster
```

12.2.10.3.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.10.3.3 SEE ALSO

- dkp detach (see page 1893) - Detach one of [cluster]

12.2.11 dkp diagnose

Generate a support bundle

```
dkp diagnose [flags]
```

12.2.11.1 Options

```
      --allow-insecure-connections     When set, do not verify TLS certs when
retrieving spec and reporting results
      --as string                      Username to impersonate for the operation.
User could be a regular user or a service account in a namespace.
      --as-group stringArray           Group to impersonate for the operation, this
 flag can be repeated to specify multiple groups. (default [])
      --as-uid string                  UID to impersonate for the operation.
      --bootstrap-kubeconfig string    Path to the kubeconfig file to use for
 requests towards an additional bootstrap cluster
      --bundle-name-prefix string      Prefix added to support bundle name created
      --cache-dir string               Default cache directory (default "/home/
runner/.kube/cache")
      --certificate-authority string   Path to a cert file for the certificate
authority
      --client-certificate string      Path to a client certificate file for TLS
      --client-key string              Path to a client key file for TLS
      --cluster string                 The name of the kubeconfig cluster to use
      --collect-without-permissions    Always generate a support bundle, even if it
some require additional permissions (default true)
      --context string                 The name of the kubeconfig context to use
      --disable-compression            If true, opt-out of response compression for
 all requests to the server
  -h, --help                           help for diagnose
      --insecure-skip-tls-verify       If true, the server's certificate will not be
checked for validity. This will make your HTTPS connections insecure
      --kubeconfig string              Path to the kubeconfig file to use for CLI
requests.
  -n, --namespace string               If present, the namespace scope for this CLI
request
      --redactors strings              Names of the additional redactors to use
(default [])
      --request-timeout string         The length of time to wait before giving up on
a single server request. Non-zero values should contain a corresponding time unit
(e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
  -s, --server string                  The address and port of the Kubernetes API
server
      --since string                   Force pod logs collectors to return logs newer
than a relative duration like 5s, 2m, or 3h.
      --since-time string              Force pod logs collectors to return logs after
a specific date (RFC3339)
      --tls-server-name string         Server name to use for server certificate
validation. If it is not provided, the hostname used to contact the server is used
      --token string                   Bearer token for authentication to the API
server
      --user string                    The name of the kubeconfig user to use
  -v, --verbose int                    Output verbosity
```

12.2.11.2 SEE ALSO

- dkp diagnose default-config (see page 1896) - Prints the default configuration of the diagnostics bundle collectors
- dkp diagnose ssh (see page 1896) - Collect node-level diagnostics data over SSH

12.2.11.3 dkp diagnose default-config

Prints the default configuration of the diagnostics bundle collectors

```
dkp diagnose default-config [flags]
```

12.2.11.3.1 Options

```
  -h, --help   help for default-config
```

12.2.11.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.11.3.3 SEE ALSO

- dkp diagnose (see page 1894) - Generate a support bundle

12.2.11.4 dkp diagnose ssh

Collect node-level diagnostics data over SSH

```
dkp diagnose ssh path/to/inventory-file.yaml [flags]
```

12.2.11.4.1 Options

```
      --bundle-name-prefix string   Prefix added to support bundle name created
  -h, --help                        help for ssh
```

```
      --redactors strings           Names of the additional redactors to use (default
[])
      --timeout duration            Timeout for collecting bundle per node (default
5m0s)
```

12.2.11.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.11.4.3 SEE ALSO

- dkp diagnose (see page 1894) - Generate a support bundle

12.2.12 dkp edit

Edit a resource on the server

12.2.12.1 Synopsis

Edit a resource from the default editor.

The edit command allows you to directly edit any API resource you can retrieve via the command-line tools. It will open the editor defined by your KUBE_EDITOR, or EDITOR environment variables, or fall back to 'vi' for Linux or 'notepad' for Windows. You can edit multiple objects, although changes are applied one at a time. The command accepts file names as well as command-line arguments, although the files you point to must be previously saved versions of resources.

```
dkp edit (RESOURCE/NAME | -f FILENAME)
```

12.2.12.2 Options

```
      --allow-missing-template-keys   If true, ignore any errors in templates when a
field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
      --config string                 Config file to use (default "/home/
runner/.kommander/config")
      --context string                The name of the kubeconfig context to use
      --field-manager string          Name of the manager used to track field
ownership. (default "kommander-cli")
  -f, --filename strings              Filename, directory, or URL to files to use to
edit the resource (default [])
  -h, --help                          help for edit
```

```
      --kubeconfig string              Path to the kubeconfig file to use for CLI
requests.
  -k, --kustomize string               Process the kustomization directory. This flag
can't be used together with -f or -R.
  -n, --namespace string               namespace of the resource (default "default")
  -o, --output string                  Output format. One of: (json, yaml, name, go-
template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-patch                   Output the patch if the resource is edited.
  -R, --recursive                      Process the directory used in -f, --filename
recursively. Useful when you want to manage related manifests organized within the
same directory.
      --request-timeout string         The length of time to wait before giving up on
a single server request. Non-zero values should contain a corresponding time unit
(e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
      --save-config                    If true, the configuration of current object
will be saved in its annotation. Otherwise, the annotation will be unchanged. This
flag is useful when you want to perform kubectl apply on this object in the future.
      --show-managed-fields            If true, keep the managedFields when printing
objects in JSON or YAML format.
      --subresource string             If specified, edit will operate on the
subresource of the requested object. Must be one of [status]. This flag is alpha and
may change in the future.
      --template string                Template string or path to template file to use
when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
      --validate string[="strict"]     Must be one of: strict (or true), warn, ignore
(or false).
                                                 "true" or "strict" will use a schema to
validate the input and fail the request if invalid. It will perform server side
validation if ServerSideFieldValidation is enabled on the api-server, but will fall
back to less reliable client-side validation if not.
                                                 "warn" will warn about unknown or
duplicate fields without blocking the request if server-side field validation is
enabled on the API server, and behave as "ignore" otherwise.
                                                 "false" or "ignore" will not perform any
schema validation, silently dropping any unknown or duplicate fields. (default
"strict")
  -v, --verbose int                    Output verbosity
      --windows-line-endings           Defaults to the line ending native to your
platform.
```

12.2.13 dkp get

Get one of [appdeployments, chart, clusters, kubeconfig, nodepools, workspaces]

12.2.13.1 Options

```
  -h, --help   help for get
```

12.2.13.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.13.3 SEE ALSO

- dkp get appdeployments (see page 1900) - Get AppDeployments from a Workspace, Project, or all Workspaces and Projects
- dkp get chart (see page 1902) - Obtain information about charts stored in the repository
- dkp get clusters (see page 1903) - Get clusters from specified Workspace
- dkp get kubeconfig (see page 1899) - Retrieve cluster kubeconfig and modify local kubeconfig file
- dkp get nodepools (see page 1901) - Get nodepools for a given cluster
- dkp get workspaces (see page 1901) - Get Workspaces

12.2.13.4 dkp get kubeconfig

Retrieve cluster kubeconfig and modify local kubeconfig file

```
dkp get kubeconfig [flags]
```

12.2.13.4.1 Options

```
      --cluster string      Kommander Cluster to get kubeconfig for
  -c, --cluster-name name   Name used to prefix the cluster and all the created
resources.
  -h, --help                help for kubeconfig
      --kubeconfig string   Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
  -n, --namespace string    If present, the namespace scope for this CLI request. (de
fault "default")
  -w, --workspace string    Name of the workspace to show clusters from
```

12.2.13.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.13.4.3 SEE ALSO

- dkp get (see page 1898) - Get one of [appdeployments, chart, clusters, kubeconfig, nodepools, workspaces]

12.2.13.5 dkp get appdeployments

Get AppDeployments from a Workspace, Project, or all Workspaces and Projects

12.2.13.5.1 Synopsis

Prints a table of the most important information about the specified resources. In case the AppDeployments are configured with cluster scoped specification, an optional CLUSTERS column (when printing in table format) will display enabled clusters for each AppDeployment.

```
dkp get appdeployments [APPDEPLOYMENT_NAME] [flags]
```

12.2.13.5.2 Options

```
  -A, --all-namespaces           If present, list the requested object(s) across all
namespaces.
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for appdeployments
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
  -o, --output string            Output format. One of: table|yaml
  -p, --project string           Name of the project to show AppDeployments from.
Requires workspace flag (workspace that the project belongs to).
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
  -w, --workspace string         Name of the workspace to show AppDeployments from
```

12.2.13.5.3 SEE ALSO

- dkp get (see page 1898) - Get one of [appdeployments, chart, clusters, kubeconfig, nodepools, workspaces]

12.2.13.6 dkp get workspaces

Get Workspaces

```
dkp get workspaces [flags]
```

12.2.13.6.1 Options

```
  -A, --all-namespaces           If present, list the requested object(s) across all
namespaces.
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for workspaces
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
  -o, --output string            Output format. One of: table|yaml
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.13.6.2 SEE ALSO

- dkp get (see page 1898) - Get one of [appdeployments, chart, clusters, kubeconfig, nodepools, workspaces]

12.2.13.7 dkp get nodepools

Get nodepools for a given cluster

```
dkp get nodepools [flags]
```

12.2.13.7.1 Options

```
  -c, --cluster-name name   Name used to prefix the cluster and all the created
resources.
  -h, --help                help for nodepools
      --kubeconfig string   Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
  -n, --namespace string    If present, the namespace scope for this CLI request. (de
fault "default")
  -v, --verbose int         Output verbosity
```

12.2.13.7.2 SEE ALSO

- dkp get (see page 1898) - Get one of [appdeployments, chart, clusters, kubeconfig, nodepools, workspaces]

12.2.13.8 dkp get chart

Obtain information about charts stored in the repository

```
dkp get chart [chartName] [chartVersion] [flags]
```

12.2.13.8.1 Options

```
  -A, --all-namespaces           If present, list the requested object(s) across all
namespaces.
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for chart
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
  -o, --output string            Output format. One of: table|yaml
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.13.8.2 SEE ALSO

- dkp get (see page 1898) - Get one of [appdeployments, chart, clusters, kubeconfig, nodepools, workspaces]

12.2.13.9 dkp get clusters

Get clusters from specified Workspace

```
dkp get clusters [CLUSTER_NAME] [flags]
```

12.2.13.9.1 Options

```
  -A, --all-namespaces           If present, list the requested object(s) across all
namespaces.
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for clusters
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
  -o, --output string            Output format. One of: table|yaml
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
  -w, --workspace string         Name of the workspace to show clusters from
```

12.2.13.9.2 SEE ALSO

- dkp get (see page 1898) - Get one of [appdeployments, chart, clusters, kubeconfig, nodepools, workspaces]

12.2.14 dkp import

Import images from an image bundle into Containerd

12.2.14.1 Options

```
  -h, --help          help for import
  -v, --verbose int   Output verbosity
```

12.2.14.2 SEE ALSO

- dkp import image-bundle (see page 1904) - Import images from image bundles into Containerd

12.2.14.3 dkp import image-bundle

Import images from image bundles into Containerd

```
dkp import image-bundle [flags]
```

12.2.14.3.1 Options

```
      --containerd-namespace string   Containerd namespace to import images into (def
ault "k8s.io")
  -h, --help                          help for image-bundle
      --image-bundle strings          Tarball containing list of images to import.
Can also be a glob pattern. (default [])
```

12.2.14.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.14.3.3 SEE ALSO

- dkp import (see page 1903) - Import images from an image bundle into Containerd

12.2.15 dkp install

Install one of [kommander]

12.2.15.1 Options

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for install
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.15.2 SEE ALSO

- dkp install kommander (see page 1905) - Install kommander

12.2.15.3 dkp install kommander

Install kommander

```
dkp install kommander [flags]
```

12.2.15.3.1 Options

```
      --airgapped                                 Enable airgapped mode.
      --bootstrap-repository string               git repository with bootstrap
definitions
      --charts-bundle stringArray                 Path to charts-bundle to upload to
chartmuseum, apart from parsing the kommander applications repository (default [])
      --disallow-charts-download                  make CLI rely solely on provided
chart bundles and do not try to download charts from the Internet
      --gitea-kommander-repository-name string    gitea kommander repository name
(default "kommander")
  -h, --help                                      help for kommander
      --init                                      Initialize default configuration
file, print it and exit without installing Kommander.
      --installer-config file                     Path to installation configuration
file
      --kommander-applications-repository string  git repository with application
definitions (default "v2.8.1")
  -o, --output string                             Output format for the
configuration file generated by --init. One of: yaml|json (default "yaml")
      --wait                                      Wait for all enabled applications
to be ready (default true)
      --wait-timeout duration                     Time to wait for all enabled
applications to be ready (30m, 1h, 2h) (default 1h0m0s)
```

12.2.15.3.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
```

```
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.15.3.3 SEE ALSO

- dkp install (see page 1904) - Install one of [kommander]

12.2.16 dkp move

Move one of [capi-resources]

12.2.16.1 Synopsis

Command "move" is deprecated, use "dkp move capi-resources" instead Move one of [capi-resources]

```
dkp move [flags]
```

12.2.16.2 Options

```
      --from-context string    Context to be used within the from-cluster's
kubeconfig file. If empty, current context will be used. (DEPRECATED: use "dkp move
capi-resources" instead)
      --from-kubeconfig file   Path to the kubeconfig for pivot's source cluster. If
unspecified, default discovery rules apply. (DEPRECATED: use "dkp move capi-
resources" instead)
  -h, --help                   help for move
  -n, --namespace string       If present, the namespace scope for this CLI request.
(default "default")
      --to-context string      Context to be used within the to-cluster's kubeconfig
file. If empty, current context will be used. (DEPRECATED: use "dkp move capi-
resources" instead)
      --to-kubeconfig file     Path to the kubeconfig for pivot's destination cluster
(DEPRECATED: use "dkp move capi-resources" instead)
      --to-namespace string    Resources are moved to this namespace in the to-
cluster. By default, the same as the from-cluster namespace.
  -v, --verbose int            Output verbosity
```

12.2.16.3 SEE ALSO

- dkp move capi-resources (see page 1907) - Move controllers and objects from one cluster to the other

12.2.16.4 dkp move capi-resources

Move controllers and objects from one cluster to the other

```
dkp move capi-resources [flags]
```

12.2.16.4.1 Options

```
      --from-context string    Context to be used within the from-cluster's
kubeconfig file. If empty, current context will be used.
      --from-kubeconfig file   Path to the kubeconfig for pivot's source cluster. If
unspecified, default discovery rules apply.
  -h, --help                   help for capi-resources
  -n, --namespace string       If present, the namespace scope for this CLI request.
(default "default")
      --to-context string      Context to be used within the to-cluster's kubeconfig
file. If empty, current context will be used.
      --to-kubeconfig file     Path to the kubeconfig for pivot's destination cluster
      --to-namespace string    Resources are moved to this namespace in the to-
cluster. By default, the same as the from-cluster namespace.
```

12.2.16.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.16.4.3 SEE ALSO

- dkp move (see page 1906) - Move one of [capi-resources]

12.2.17 dkp open

Open one of [dashboard]

12.2.17.1 Options

```
      --config string           Config file to use (default "/home/
runner/.kommander/config")
      --context string          The name of the kubeconfig context to use
```

```
  -h, --help                     help for open
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.17.2 SEE ALSO

- dkp open dashboard (see page 1908) - Open DKP UI in your browser

12.2.17.3 dkp open dashboard

Open DKP UI in your browser

```
dkp open dashboard [flags]
```

12.2.17.3.1 Options

```
  -h, --help   help for dashboard
```

12.2.17.3.2 Options inherited from parent commands

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.17.3.3 SEE ALSO

- dkp open (see page 1907) - Open one of [dashboard]

12.2.18 dkp push

Push one of [bundle, chart, chart-bundle, image-bundle]

12.2.18.1 Options

```
  -h, --help   help for push
```

12.2.18.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.18.3 SEE ALSO

- dkp push bundle (see page 1910) - Push from bundles into an existing OCI registry
- dkp push chart (see page 1910) - Upload charts to the repository
- dkp push chart-bundle (see page 1909) - Upload chart bundles to the repository

12.2.18.4 dkp push chart-bundle

Upload chart bundles to the repository

```
dkp push chart-bundle [chartsTarball]... [flags]
```

12.2.18.4.1 Options

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for chart-bundle
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.18.4.2 SEE ALSO

- dkp push (see page 1908) - Push one of [bundle, chart, chart-bundle, image-bundle]

12.2.18.5 dkp push chart

Upload charts to the repository

```
dkp push chart [chartTarball]... [flags]
```

12.2.18.5.1 Options

```
      --config string            Config file to use (default "/home/
runner/.kommander/config")
      --context string           The name of the kubeconfig context to use
  -h, --help                     help for chart
      --kubeconfig string        Path to the kubeconfig file to use for CLI requests.
      --request-timeout string   The length of time to wait before giving up on a
single server request. Non-zero values should contain a corresponding time unit (e.g.
1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int              Output verbosity
```

12.2.18.5.2 SEE ALSO

- dkp push (see page 1908) - Push one of [bundle, chart, chart-bundle, image-bundle]

12.2.18.6 dkp push bundle

Push from bundles into an existing OCI registry

```
dkp push bundle [flags]
```

12.2.18.6.1 Options

```
      --bundle strings                          Tarball containing list of images to
push. Can also be a glob pattern. (default [])
      --ecr-lifecycle-policy-file string        File containing ECR lifecycle policy
for newly created repositories (only applies if target registry is hosted on ECR,
ignored otherwise)
  -h, --help                                    help for bundle
      --image-push-concurrency int              Image push concurrency (default 1)
      --on-existing-tag string                  how to handle existing tags: one of
"overwrite", "error", or "skip" (default "overwrite")
```

```
      --to-registry string                     Registry to push images to. TLS
verification will be skipped when using an http:// registry.
      --to-registry-ca-cert-file string        CA certificate file used to verify TLS
verification of registry to push images to
      --to-registry-insecure-skip-tls-verify   Skip TLS verification of registry to
push images to (also use for non-TLS http registries)
      --to-registry-password string            Password to use to log in to
destination registry
      --to-registry-username string            Username to use to log in to
destination registry
  -v, --verbose int                            Output verbosity
```

12.2.18.6.2 SEE ALSO

- dkp push (see page 1908) - Push one of [bundle, chart, chart-bundle, image-bundle]

12.2.19 dkp scale

Scale one of [nodepool]

12.2.19.1 Options

```
  -h, --help          help for scale
  -v, --verbose int   Output verbosity
```

12.2.19.2 SEE ALSO

- dkp scale nodepool (see page 1911) - Scale a nodepool of a given cluster to the number of replicas

12.2.19.3 dkp scale nodepool

Scale a nodepool of a given cluster to the number of replicas

```
dkp scale nodepool name [flags]
```

12.2.19.3.1 Options

```
  -c, --cluster-name name         Name used to prefix the cluster and all the created
resources.
  -h, --help                      help for nodepool
```

```
      --kubeconfig string         Path to the kubeconfig for the management cluster.
If unspecified, default discovery rules apply.
  -n, --namespace string          If present, the namespace scope for this CLI
request. (default "default")
      --nodes-to-delete strings   A list of node names to mark for deletion when
scaling down a node pool. If left empty, the nodes to delete will be selected at
random. (default [])
      --replicas int              The new desired number of replicas.
```

12.2.19.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.19.3.3 SEE ALSO

- dkp scale (see page 1911) - Scale one of [nodepool]

12.2.20 dkp serve

Serve image or Helm chart bundles from an OCI registry

12.2.20.1 Options

```
  -h, --help          help for serve
  -v, --verbose int   Output verbosity
```

12.2.20.2 SEE ALSO

- dkp serve bundle (see page 1912) - Serve an OCI registry from previously created bundles

12.2.20.3 dkp serve bundle

Serve an OCI registry from previously created bundles

```
dkp serve bundle [flags]
```

12.2.20.3.1 Options

```
      --bundle strings                Bundle to serve. Can also be a glob pattern.
(default [])
  -h, --help                          help for bundle
      --listen-address string         Address to listen on (default "127.0.0.1")
      --listen-port uint16            Port to listen on (0 means use any free port)
      --tls-cert-file string          TLS certificate file
      --tls-private-key-file string   TLS private key file
```

12.2.20.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.20.3.3 SEE ALSO

- dkp serve (see page 1912) - Serve image or Helm chart bundles from an OCI registry

12.2.21 dkp update

Update one of [bootstrap (cluster), controlplane, nodepool]

12.2.21.1 Options

```
  -h, --help          help for update
  -v, --verbose int   Output verbosity
```

12.2.21.2 SEE ALSO

- dkp update bootstrap (see page 1928) - Update bootstrap cluster
- dkp update controlplane (see page 1920) - Update a Kubernetes cluster control plane, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]
- dkp update nodepool (see page 1913) - Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.3 dkp update nodepool

Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.3.1 Options

```
  -h, --help   help for nodepool
```

12.2.21.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.3.3 SEE ALSO

- dkp update (see page 1913) - Update one of [bootstrap (cluster), controlplane, nodepool]
- dkp update nodepool aws (see page 1915) - Update a Konvoy cluster node pool in AWS
- dkp update nodepool azure (see page 1914) - Update a Konvoy cluster node pool in Azure
- dkp update nodepool eks (see page 1917) - Update a Konvoy cluster node pool in EKS
- dkp update nodepool gcp (see page 1919) - Update a Konvoy cluster node pool in GCP
- dkp update nodepool preprovisioned (see page 1916) - Update a Konvoy cluster node pool in Preprovisioned
- dkp update nodepool vcd (see page 1919) - Update a Konvoy cluster node pool in Vcd
- dkp update nodepool vsphere (see page 1918) - Update a Konvoy cluster node pool in vSphere

12.2.21.3.4 dkp update nodepool azure

Update a Konvoy cluster node pool in Azure

```
dkp update nodepool azure [flags]
```

12.2.21.3.4.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
      --compute-gallery-id string   Compute Gallery ID of a custom image, e.g., '/
subscriptions/<subscription id>/resourceGroups/<resource group name>/providers/
Microsoft.Compute/galleries/<gallery name>/images/<image definition name>/versions/
<version id>' (replacing placeholders with the values used when creating the image)
  -h, --help                        help for azure
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
```

```
      --machine-size string         Worker machine size (ex. 'Standard_D2s_v3')
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --plan-offer string           The offer for a Marketplace image or a custom
image sourced from a Marketplace image requiring Plan information.
      --plan-publisher string       The publisher for a Marketplace image or a custom
image sourced from a Marketplace image requiring Plan information.
      --plan-sku string             The SKU for a Marketplace image or a custom image
sourced from a Marketplace image requiring Plan information.
      --timeout duration            The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

12.2.21.3.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.3.4.3 SEE ALSO

- dkp update nodepool (see page 1913) - Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.3.5 dkp update nodepool aws

Update a Konvoy cluster node pool in AWS

```
dkp update nodepool aws [flags]
```

12.2.21.3.5.1 Options

```
      --ami string                  AMI ID to use for machines
      --ami-base-os string          Base OS used in search of AMIs. Examples:
'centos-7', 'ubuntu-18.04', 'ubuntu-20.04'
      --ami-format string           Query string used in search of AMIs. Example:
When --ami-base-os='rhel8.4', then the string 'prefix-{{.BaseOS}}-?{{.K8sVersion}}-*'
 matches any AMIs with the Name 'prefix-rhel8.4-1.28.7
      --ami-owner string            ID of AWS account used in search of AMIs
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for aws
      --instance-type string        Instance type to use for node pool machines
```

```
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --timeout duration            The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

12.2.21.3.5.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.3.5.3 SEE ALSO

- dkp update nodepool (see page 1913) - Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.3.6 dkp update nodepool preprovisioned

Update a Konvoy cluster node pool in Preprovisioned

```
dkp update nodepool preprovisioned [flags]
```

12.2.21.3.6.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for preprovisioned
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --timeout duration            The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

12.2.21.3.6.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.3.6.3 SEE ALSO

- dkp update nodepool (see page 1913) - Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.3.7 dkp update nodepool eks

Update a Konvoy cluster node pool in EKS

```
dkp update nodepool eks [flags]
```

12.2.21.3.7.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for eks
      --instance-type string        Instance type to use for node pool machines
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --timeout duration            The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

12.2.21.3.7.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.3.7.3 SEE ALSO

- dkp update nodepool (see page 1913) - Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.3.8 dkp update nodepool vsphere

Update a Konvoy cluster node pool in vSphere

```
dkp update nodepool vsphere [flags]
```

12.2.21.3.8.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
      --cpus int                    The number of virtual processors in a virtual
machine.
      --disk-size int               The size of a virtual machine's disk, in GB.
  -h, --help                        help for vsphere
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
      --memory int                  The size of a virtual machine's memory, in GB.
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --timeout duration            The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --use-context string          Use a specific context in a kubeconfig file.
      --vm-template string          The virtual machine template to use for a virtual
machine.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

12.2.21.3.8.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.3.8.3 SEE ALSO

- dkp update nodepool (see page 1913) - Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.3.9 dkp update nodepool gcp

Update a Konvoy cluster node pool in GCP

```
dkp update nodepool gcp [flags]
```

12.2.21.3.9.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for gcp
      --image string                Full reference to an image to use for all nodes
(set either this or --image-family) (ex. 'projects/my-project/global/images/konvoy-
ubuntu-2004-1-99-99-1234567890')
      --image-family string         Full reference to an image family to use for all
nodes (set either this or --image) (ex. 'projects/my-project/global/images/family/
konvoy-ubuntu-2004-{{.K8sVersion}}')
      --instance-type string        Worker machine instance type (ex. "n2-standard-8")
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --timeout duration            The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

12.2.21.3.9.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.3.9.3 SEE ALSO

- dkp update nodepool (see page 1913) - Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.3.10 dkp update nodepool vcd

Update a Konvoy cluster node pool in Vcd

```
dkp update nodepool vcd [flags]
```

12.2.21.3.10.1 Options

```
      --catalog string              Catalog hosting the virtual machine vApp
template.
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for vcd
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --timeout duration            The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --use-context string          Use a specific context in a kubeconfig file.
      --vapp-template string        The vApp template to use for a virtual machine.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

12.2.21.3.10.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.3.10.3 SEE ALSO

- dkp update nodepool (see page 1913) - Upate a Kubernetes cluster node pool, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.4 dkp update controlplane

Update a Kubernetes cluster control plane, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.4.1 Options

```
  -h, --help   help for controlplane
```

12.2.21.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.4.3 SEE ALSO

- dkp update (see page 1913) - Update one of [bootstrap (cluster), controlplane, nodepool]
- dkp update controlplane aks (see page 1927) - Update a Konvoy cluster control plane in AKS
- dkp update controlplane aws (see page 1922) - Update a Konvoy cluster control plane in AWS
- dkp update controlplane azure (see page 1923) - Update a Konvoy cluster control plane in Azure
- dkp update controlplane eks (see page 1924) - Update a Konvoy cluster control plane in EKS
- dkp update controlplane gcp (see page 1925) - Update a Konvoy cluster control plane in GCP
- dkp update controlplane preprovisioned (see page 1921) - Update a Konvoy cluster control plane in Preprovisioned
- dkp update controlplane vcd (see page 1926) - Update a Konvoy cluster control plane in Vcd
- dkp update controlplane vsphere (see page 1925) - Update a Konvoy cluster control plane in vSphere

12.2.21.4.4 dkp update controlplane preprovisioned

Update a Konvoy cluster control plane in Preprovisioned

```
dkp update controlplane preprovisioned [flags]
```

12.2.21.4.4.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for preprovisioned
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --timeout duration            The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

12.2.21.4.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.4.4.3 SEE ALSO

- dkp update controlplane (see page 1920) - Update a Kubernetes cluster control plane, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.4.5 dkp update controlplane aws

Update a Konvoy cluster control plane in AWS

```
dkp update controlplane aws [flags]
```

12.2.21.4.5.1 Options

```
      --ami string                  AMI ID to use for machines
      --ami-base-os string          Base OS used in search of AMIs. Examples:
'centos-7', 'ubuntu-18.04', 'ubuntu-20.04'
      --ami-format string           Query string used in search of AMIs. Example:
When --ami-base-os='rhel8.4', then the string 'prefix-{{.BaseOS}}-?{{.K8sVersion}}-*'
 matches any AMIs with the Name 'prefix-rhel8.4-1.28.7
      --ami-owner string            ID of AWS account used in search of AMIs
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for aws
      --instance-type string        Instance type to use for control plane machines
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --timeout duration            The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

12.2.21.4.5.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.4.5.3 SEE ALSO

- dkp update controlplane (see page 1920) - Update a Kubernetes cluster control plane, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.4.6 dkp update controlplane azure

Update a Konvoy cluster control plane in Azure

```
dkp update controlplane azure [flags]
```

12.2.21.4.6.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
      --compute-gallery-id string   Compute Gallery ID of a custom image, e.g., '/
subscriptions/<subscription id>/resourceGroups/<resource group name>/providers/
Microsoft.Compute/galleries/<gallery name>/images/<image definition name>/versions/
<version id>' (replacing placeholders with the values used when creating the image)
  -h, --help                        help for azure
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
      --machine-size string         Worker machine size (ex. 'Standard_D2s_v3')
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --plan-offer string           The offer for a Marketplace image or a custom
image sourced from a Marketplace image requiring Plan information.
      --plan-publisher string       The publisher for a Marketplace image or a custom
image sourced from a Marketplace image requiring Plan information.
      --plan-sku string             The SKU for a Marketplace image or a custom image
sourced from a Marketplace image requiring Plan information.
      --timeout duration            The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

12.2.21.4.6.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.4.6.3 SEE ALSO

- dkp update controlplane (see page 1920) - Update a Kubernetes cluster control plane, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.4.7 dkp update controlplane eks

Update a Konvoy cluster control plane in EKS

```
dkp update controlplane eks [flags]
```

12.2.21.4.7.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for eks
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --timeout duration            The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

12.2.21.4.7.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.4.7.3 SEE ALSO

- dkp update controlplane (see page 1920) - Update a Kubernetes cluster control plane, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.4.8 dkp update controlplane vsphere

Update a Konvoy cluster control plane in vSphere

```
dkp update controlplane vsphere [flags]
```

12.2.21.4.8.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
      --cpus int                    The number of virtual processors in a virtual
machine.
      --disk-size int               The size of a virtual machine's disk, in GB.
  -h, --help                        help for vsphere
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
      --memory int                  The size of a virtual machine's memory, in GB.
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --timeout duration            The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --use-context string          Use a specific context in a kubeconfig file.
      --vm-template string          The virtual machine template to use for a virtual
machine.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

12.2.21.4.8.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.4.8.3 SEE ALSO

- dkp update controlplane (see page 1920) - Update a Kubernetes cluster control plane, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.4.9 dkp update controlplane gcp

Update a Konvoy cluster control plane in GCP

```
dkp update controlplane gcp [flags]
```

12.2.21.4.9.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for gcp
      --image string                Full reference to an image to use for all nodes
(set either this or --image-family) (ex. 'projects/my-project/global/images/konvoy-
ubuntu-2004-1-99-99-1234567890')
      --image-family string         Full reference to an image family to use for all
nodes (set either this or --image) (ex. 'projects/my-project/global/images/family/
konvoy-ubuntu-2004-{{.K8sVersion}}')
      --instance-type string        Control Plane machine instance type (ex. "n2-
standard-4")
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --timeout duration            The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

12.2.21.4.9.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.4.9.3 SEE ALSO

- dkp update controlplane (see page 1920) - Update a Kubernetes cluster control plane, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.4.10 dkp update controlplane vcd

Update a Konvoy cluster control plane in Vcd

```
dkp update controlplane vcd [flags]
```

12.2.21.4.10.1 Options

```
      --catalog string              Catalog hosting the virtual machine vApp
template.
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for vcd
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --timeout duration            The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --use-context string          Use a specific context in a kubeconfig file.
      --vapp-template string        The vApp template to use for a virtual machine.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

12.2.21.4.10.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.4.10.3 SEE ALSO

- dkp update controlplane (see page 1920) - Update a Kubernetes cluster control plane, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.4.11 dkp update controlplane aks

Update a Konvoy cluster control plane in AKS

```
dkp update controlplane aks [flags]
```

12.2.21.4.11.1 Options

```
  -c, --cluster-name name           Name used to prefix the cluster and all the
created resources.
  -h, --help                        help for aks
```

```
      --kubeconfig string           Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --kubernetes-version string   Kubernetes version
  -n, --namespace string            If present, the namespace scope for this CLI
request. (default "default")
      --timeout duration            The length of time to wait before giving up. Zero
means wait forever (e.g. 300s, 30m, 3h). (default 30m0s)
      --use-context string          Use a specific context in a kubeconfig file.
      --wait                        If true, wait for operations to complete before
returning. (default true)
```

12.2.21.4.11.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.4.11.3 SEE ALSO

- dkp update controlplane (see page 1920) - Update a Kubernetes cluster control plane, one of [aks, aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.21.5 dkp update bootstrap

Update bootstrap cluster

12.2.21.5.1 Options

```
  -h, --help   help for bootstrap
```

12.2.21.5.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.5.3 SEE ALSO

- dkp update (see page 1913) - Update one of [bootstrap (cluster), controlplane, nodepool]
- dkp update bootstrap credentials (see page 1929) - Update credentials in the cluster

12.2.21.5.4 dkp update bootstrap credentials

Update credentials in the cluster

12.2.21.5.4.1 Options

```
  -h, --help   help for credentials
```

12.2.21.5.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.21.5.4.3 SEE ALSO

- dkp update bootstrap (see page 1928) - Update bootstrap cluster
- dkp update bootstrap credentials aws (see page 1929) - Update AWS credentials in the cluster and restart CAPA controllers
- dkp update bootstrap credentials gcp (see page 1930) - Update GCP credentials in the cluster and restart CAPG controllers
- dkp update bootstrap credentials vsphere (see page 1930) - Update VSphere credentials in the cluster and restart CAPV controllers

12.2.21.5.4.4 dkp update bootstrap credentials aws

Update AWS credentials in the cluster and restart CAPA controllers

```
dkp update bootstrap credentials aws [flags]
```

Options

```
      --context string      The name of the kubeconfig context to use
  -h, --help                help for aws
      --kubeconfig string   Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
      --print-only          Print the credentials and exit the function. Without
modifying cluster
```

Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

SEE ALSO

- dkp update bootstrap credentials (see page 1929) - Update credentials in the cluster

12.2.21.5.4.5 dkp update bootstrap credentials gcp

Update GCP credentials in the cluster and restart CAPG controllers

```
dkp update bootstrap credentials gcp [flags]
```

Options

```
      --context string      The name of the kubeconfig context to use
  -h, --help                help for gcp
      --kubeconfig string   Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
      --print-only          Print the credentials and exit the function. Without
modifying cluster
```

Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

SEE ALSO

- dkp update bootstrap credentials (see page 1929) - Update credentials in the cluster

12.2.21.5.4.6 dkp update bootstrap credentials vsphere

Update VSphere credentials in the cluster and restart CAPV controllers

```
dkp update bootstrap credentials vsphere [flags]
```

Options

```
      --context string      The name of the kubeconfig context to use
  -h, --help                help for vsphere
      --kubeconfig string   Path to the kubeconfig for the management cluster. If
unspecified, default discovery rules apply.
      --print-only          Print the credentials and exit the function. Without
modifying cluster
```

Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

SEE ALSO

- dkp update bootstrap credentials (see page 1929) - Update credentials in the cluster

12.2.22 dkp upgrade

Upgrade one of [addons, capi-components, catalogapp, cluster, kommander, workspace]

12.2.22.1 Options

```
  -h, --help   help for upgrade
```

12.2.22.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.22.3 SEE ALSO

- dkp upgrade addons (see page 1935) - Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]
- dkp upgrade capi-components (see page 1933) - Upgrade the CAPI components in the cluster
- dkp upgrade catalogapp (see page 1933) - Upgrade a Catalog Application to a newer version
- dkp upgrade kommander (see page 1932) - Upgrade the Kommander version of the targeted cluster

- dkp upgrade workspace (see page 1934) - Upgrade all platform applications in the given workspace and its projects to the same version as platform applications running on the management cluster

12.2.22.4 dkp upgrade kommander

Upgrade the Kommander version of the targeted cluster

12.2.22.4.1 Synopsis

Upgrades all Kommander components and platform applications running on the targeted cluster. No attached clusters and applications running on them are affected by this action.

```
dkp upgrade kommander [flags]
```

12.2.22.4.2 Options

```
      --charts-bundle stringArray                  Path to charts-bundle to upload to
chartmuseum, apart from parsing the kommander applications repository (default [])
      --config string                              Config file to use (default "/
home/runner/.kommander/config")
      --context string                             The name of the kubeconfig context
to use
      --core-app-timeout duration                  Timeout to wait for upgrade of
each kommander core application (default 20m0s)
      --disable-appdeployments strings             List of AppDeployments to be
disabled during upgrade (default [])
      --disallow-charts-download                   make CLI rely solely on provided
chart bundles and do not try to download charts from the Internet
      --gitea-kommander-repository-name string     gitea kommander repository name
(default "kommander")
  -h, --help                                       help for kommander
      --kommander-applications-repository string   git repository with application
definitions (default "v2.8.1")
      --kommander-charts-version string            Kommander helm charts version to
download. Default: download all available versions
      --kubeconfig string                          Path to the kubeconfig file to use
for CLI requests.
      --platform-apps-timeout duration             Timeout to wait for upgrade of the
set of platform applications (default 30m0s)
      --request-timeout string                     The length of time to wait before
giving up on a single server request. Non-zero values should contain a corresponding
time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int                                Output verbosity
```

12.2.22.4.3 SEE ALSO

- dkp upgrade (see page 1931) - Upgrade one of [addons, capi-components, catalogapp, cluster, kommander, workspace]

12.2.22.5 dkp upgrade capi-components

Upgrade the CAPI components in the cluster

```
dkp upgrade capi-components [flags]
```

12.2.22.5.1 Options

```
      --aws-service-endpoints string     Custom AWS service endpoints in a semi-colon
separated format: ${SigningRegion1}:${ServiceID1}=${URL},${ServiceID2}=${URL};$
{SigningRegion2}...
  -h, --help                             help for capi-components
      --http-proxy string                HTTP proxy for CAPI controllers
      --https-proxy string               HTTPS proxy for CAPI controllers
      --kubeconfig string                Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
      --no-proxy strings                 No Proxy list for CAPI controllers (default
 [])
      --timeout duration                 The length of time to wait before giving up.
Zero means wait forever (e.g. 300s, 30m, 3h). (default 10m0s)
  -v, --verbose int                      Output verbosity
      --wait                             If true, wait for operations to complete
before returning. (default true)
      --with-aws-bootstrap-credentials   Set true to use AWS bootstrap credentials
from your environment. When false, the instance profile of the EC2 instance where the
CAPA controller is scheduled on will be used instead.
      --with-gcp-bootstrap-credentials   Set true to use GCP bootstrap credentials
from your environment. When false, the service account of the VM instance where the
CAPG controller is scheduled on will be used instead.
```

12.2.22.5.2 SEE ALSO

- dkp upgrade (see page 1931) - Upgrade one of [addons, capi-components, catalogapp, cluster, kommander, workspace]

12.2.22.6 dkp upgrade catalogapp

Upgrade a Catalog Application to a newer version

```
dkp upgrade catalogapp CATALOGAPP_NAME --to-version VERSION [--workspace WORKSPACE |
--project PROJECT] [flags]
```

12.2.22.6.1 Options

```
      --config string                    Config file to use (default "/home/
runner/.kommander/config")
      --context string                   The name of the kubeconfig context to use
      --core-app-timeout duration        Timeout to wait for upgrade of each
kommander core application (default 20m0s)
      --disable-appdeployments strings   List of AppDeployments to be disabled during
upgrade (default [])
  -h, --help                             help for catalogapp
      --kubeconfig string                Path to the kubeconfig file to use for CLI
requests.
      --platform-apps-timeout duration   Timeout to wait for upgrade of the set of
platform applications (default 30m0s)
      --project string                   Name of the Project to upgrade the Catalog
App in
      --request-timeout string           The length of time to wait before giving up
on a single server request. Non-zero values should contain a corresponding time unit
(e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
      --to-version string                Version the Catalog App should be upgraded
to
  -v, --verbose int                      Output verbosity
  -w, --workspace string                 Name of the Workspace to upgrade the Catalog
App in
```

12.2.22.6.2 SEE ALSO

- dkp upgrade (see page 1931) - Upgrade one of [addons, capi-components, catalogapp, cluster, kommander, workspace]

12.2.22.7 dkp upgrade workspace

Upgrade all platform applications in the given workspace and its projects to the same version as platform applications running on the management cluster

```
dkp upgrade workspace WORKSPACE_NAME [--dry-run] [flags]
```

12.2.22.7.1 Options

```
      --config string                    Config file to use (default "/home/
runner/.kommander/config")
      --context string                   The name of the kubeconfig context to use
      --core-app-timeout duration        Timeout to wait for upgrade of each
kommander core application (default 20m0s)
      --disable-appdeployments strings   List of AppDeployments to be disabled during
upgrade (default [])
      --dry-run                          Do not upgrade, just list the AppDeployments
that would be upgraded
  -h, --help                             help for workspace
      --kubeconfig string                Path to the kubeconfig file to use for CLI
requests.
      --platform-apps-timeout duration   Timeout to wait for upgrade of the set of
platform applications (default 30m0s)
      --request-timeout string           The length of time to wait before giving up
on a single server request. Non-zero values should contain a corresponding time unit
(e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
  -v, --verbose int                      Output verbosity
```

12.2.22.7.2 SEE ALSO

- dkp upgrade (see page 1931) - Upgrade one of [addons, capi-components, catalogapp, cluster, kommander, workspace]

12.2.22.8 dkp upgrade addons

Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.22.8.1 Options

```
  -h, --help          help for addons
  -v, --verbose int   Output verbosity
```

12.2.22.8.2 SEE ALSO

- dkp upgrade (see page 1931) - Upgrade one of [addons, capi-components, catalogapp, cluster, kommander, workspace]
- dkp upgrade addons aws (see page 1936) - Upgrade the core Addons in a AWS cluster
- dkp upgrade addons azure (see page 1937) - Upgrade the core Addons in a Azure cluster
- dkp upgrade addons eks (see page 1939) - Upgrade the core Addons in a EKS cluster
- dkp upgrade addons gcp (see page 1941) - Upgrade the core Addons in a GCP cluster

- dkp upgrade addons preprovisioned (see page 1938) - Upgrade the core Addons in a Preprovisioned cluster
- dkp upgrade addons vcd (see page 1942) - Upgrade the core Addons in a Vcd cluster
- dkp upgrade addons vsphere (see page 1940) - Upgrade the core Addons in a vSphere cluster

12.2.22.8.3 dkp upgrade addons aws

Upgrade the core Addons in a AWS cluster

```
dkp upgrade addons aws [flags]
```

12.2.22.8.3.1 Options

```
      --allow-missing-template-keys   If true, ignore any errors in templates when a
field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
  -c, --cluster-name name             Name used to prefix the cluster and all the
created resources.
      --dry-run                       Only print the objects that would be created,
without creating them.
  -h, --help                          help for aws
      --kubeconfig string             Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
  -o, --output string                 Output format. One of: (json, yaml, name, go-
template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string       Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --show-managed-fields           If true, keep the managedFields when printing
objects in JSON or YAML format.
      --template string               Template string or path to template file to use
when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
```

12.2.22.8.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.22.8.3.3 SEE ALSO

- dkp upgrade addons (see page 1935) - Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.22.8.4 dkp upgrade addons azure

Upgrade the core Addons in a Azure cluster

```
dkp upgrade addons azure [flags]
```

12.2.22.8.4.1 Options

```
      --allow-missing-template-keys   If true, ignore any errors in templates when a
field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
  -c, --cluster-name name             Name used to prefix the cluster and all the
created resources.
      --dry-run                       Only print the objects that would be created,
without creating them.
  -h, --help                          help for azure
      --kubeconfig string             Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
  -o, --output string                 Output format. One of: (json, yaml, name, go-
template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string       Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --show-managed-fields           If true, keep the managedFields when printing
objects in JSON or YAML format.
      --template string               Template string or path to template file to use
when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
```

12.2.22.8.4.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.22.8.4.3 SEE ALSO

- dkp upgrade addons (see page 1935) - Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.22.8.5 dkp upgrade addons preprovisioned

Upgrade the core Addons in a Preprovisioned cluster

```
dkp upgrade addons preprovisioned [flags]
```

12.2.22.8.5.1 Options

```
      --allow-missing-template-keys   If true, ignore any errors in templates when a
field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
  -c, --cluster-name name             Name used to prefix the cluster and all the
created resources.
      --dry-run                       Only print the objects that would be created,
without creating them.
  -h, --help                          help for preprovisioned
      --kubeconfig string             Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
  -o, --output string                 Output format. One of: (json, yaml, name, go-
template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string       Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --show-managed-fields           If true, keep the managedFields when printing
objects in JSON or YAML format.
      --template string               Template string or path to template file to use
when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
```

12.2.22.8.5.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.22.8.5.3 SEE ALSO

- dkp upgrade addons (see page 1935) - Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.22.8.6 dkp upgrade addons eks

Upgrade the core Addons in a EKS cluster

```
dkp upgrade addons eks [flags]
```

12.2.22.8.6.1 Options

```
      --allow-missing-template-keys   If true, ignore any errors in templates when a
field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
  -c, --cluster-name name             Name used to prefix the cluster and all the
created resources.
      --dry-run                       Only print the objects that would be created,
without creating them.
  -h, --help                          help for eks
      --kubeconfig string             Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
  -o, --output string                 Output format. One of: (json, yaml, name, go-
template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string       Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --show-managed-fields           If true, keep the managedFields when printing
objects in JSON or YAML format.
      --template string               Template string or path to template file to use
when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
```

12.2.22.8.6.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.22.8.6.3 SEE ALSO

- dkp upgrade addons (see page 1935) - Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.22.8.7 dkp upgrade addons vsphere

Upgrade the core Addons in a vSphere cluster

```
dkp upgrade addons vsphere [flags]
```

12.2.22.8.7.1 Options

```
      --allow-missing-template-keys   If true, ignore any errors in templates when a
field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
  -c, --cluster-name name             Name used to prefix the cluster and all the
created resources.
      --dry-run                       Only print the objects that would be created,
without creating them.
  -h, --help                          help for vsphere
      --kubeconfig string             Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
  -o, --output string                 Output format. One of: (json, yaml, name, go-
template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string       Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --show-managed-fields           If true, keep the managedFields when printing
objects in JSON or YAML format.
      --template string               Template string or path to template file to use
when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
```

12.2.22.8.7.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.22.8.7.3 SEE ALSO

- dkp upgrade addons (see page 1935) - Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.22.8.8 dkp upgrade addons gcp

Upgrade the core Addons in a GCP cluster

```
dkp upgrade addons gcp [flags]
```

12.2.22.8.8.1 Options

```
      --allow-missing-template-keys   If true, ignore any errors in templates when a
field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
  -c, --cluster-name name             Name used to prefix the cluster and all the
created resources.
      --dry-run                       Only print the objects that would be created,
without creating them.
  -h, --help                          help for gcp
      --kubeconfig string             Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
  -o, --output string                 Output format. One of: (json, yaml, name, go-
template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string       Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --show-managed-fields           If true, keep the managedFields when printing
objects in JSON or YAML format.
      --template string               Template string or path to template file to use
when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
```

12.2.22.8.8.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.22.8.8.3 SEE ALSO

- dkp upgrade addons (see page 1935) - Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.22.8.9 dkp upgrade addons vcd

Upgrade the core Addons in a Vcd cluster

```
dkp upgrade addons vcd [flags]
```

12.2.22.8.9.1 Options

```
      --allow-missing-template-keys   If true, ignore any errors in templates when a
field or map key is missing in the template. Only applies to golang and jsonpath
output formats. (default true)
  -c, --cluster-name name             Name used to prefix the cluster and all the
created resources.
      --dry-run                       Only print the objects that would be created,
without creating them.
  -h, --help                          help for vcd
      --kubeconfig string             Path to the kubeconfig for the management
cluster. If unspecified, default discovery rules apply.
  -n, --namespace string              If present, the namespace scope for this CLI
request. (default "default")
  -o, --output string                 Output format. One of: (json, yaml, name, go-
template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json,
jsonpath-file).
      --output-directory string       Used with --output=json|yaml. The directory
where to output resources to files. The directory must already exist.
      --show-managed-fields           If true, keep the managedFields when printing
objects in JSON or YAML format.
      --template string               Template string or path to template file to use
when -o=go-template, -o=go-template-file. The template format is golang templates
[http://golang.org/pkg/text/template/#pkg-overview].
```

12.2.22.8.9.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.22.8.9.3 SEE ALSO

- dkp upgrade addons (see page 1935) - Upgrade the core Addons in a cluster, one of [aws, azure, eks, gcp, preprovisioned, vcd, vsphere]

12.2.23 dkp version

Print version information

12.2.23.1 Synopsis

Print version information

```
dkp version [flags]
```

12.2.23.2 Options

```
  -h, --help            help for version
      --long            If true, print additional version information.
  -o, --output string   One of 'yaml' or 'json'.
```

12.2.23.3 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.24 dkp upload

Collect and upload telemetry information

12.2.24.1 Options

```
  -h, --help          help for upload
  -v, --verbose int   Output verbosity
```

12.2.24.2 SEE ALSO

- dkp upload telemetry (see page 1944) - Upload telemetry information to a destination

12.2.24.3 dkp upload telemetry

Upload telemetry information to a destination

```
dkp upload telemetry [flags]
```

12.2.24.3.1 Options

```
      --as string                      Username to impersonate for the operation.
User could be a regular user or a service account in a namespace.
      --as-group stringArray           Group to impersonate for the operation, this
 flag can be repeated to specify multiple groups. (default [])
      --as-uid string                  UID to impersonate for the operation.
      --bundle-profile string          available profiles are: [telemetry,
diagnostics] (default "telemetry")
      --cache-dir string               Default cache directory (default "/home/
runner/.kube/cache")
      --certificate-authority string   Path to a cert file for the certificate
authority
      --client-certificate string      Path to a client certificate file for TLS
      --client-key string              Path to a client key file for TLS
      --cluster string                 The name of the kubeconfig cluster to use
      --context string                 The name of the kubeconfig context to use
      --disable-compression            If true, opt-out of response compression for
 all requests to the server
  -h, --help                           help for telemetry
      --insecure-skip-tls-verify       If true, the server's certificate will not be
checked for validity. This will make your HTTPS connections insecure
      --kubeconfig string              Path to the kubeconfig file to use for CLI
requests.
  -n, --namespace string               If present, the namespace scope for this CLI
request
      --request-timeout string         The length of time to wait before giving up on
a single server request. Non-zero values should contain a corresponding time unit
(e.g. 1s, 2m, 3h). A value of zero means don't timeout requests.
  -s, --server string                  The address and port of the Kubernetes API
server
      --tls-server-name string         Server name to use for server certificate
validation. If it is not provided, the hostname used to contact the server is used
      --token string                   Bearer token for authentication to the API
server
```

```
      --user string                    The name of the kubeconfig user to use
```

12.2.24.3.2 Options inherited from parent commands

```
  -v, --verbose int   Output verbosity
```

12.2.24.3.3 SEE ALSO

- dkp upload (see page 1943) - Collect and upload telemetry information

13 Release Notes

View release-specific information for DKP

Download DKP

You must be a registered user and logged on to the support portal to download this product. New customers must contact their sales representative or sales@d2iq.com[1417] before attempting to download or install DKP.

Release Notes for specific releases:

- DKP 2.8.0 Release Notes (see page 1946)
- DKP 2.8.1 Release Notes (see page 1960)

13.1 DKP 2.8.0 Release Notes

DKP® version 2.8.0 was released on May 22, 2024.

Download DKP

You must be a registered user and logged on to the support portal to download this product. New customers must contact their sales representative or sales@d2iq.com[1418] before attempting to download or install DKP.

13.1.1 Release Summary

Welcome to D2iQ Kubernetes Platform (DKP) 2.8.0! This release focuses on improved features for multi-tenancy for scalable and resilient fleet management. Below, you will find information about the following:

- DKP 2.8.0 Features and Enhancements (see page 1947)
- DKP 2.8.0 Supported Kubernetes Versions (see page 1948)
- DKP 2.8.0 Kubernetes Major Updates and Deprecations (see page 1949)
- DKP 2.8.0 Components and Applications (see page 1950)

1417 mailto:sales@d2iq.com
1418 mailto:sales@d2iq.com

- DKP 2.8.0 Known Issues and Limitations (see page 1958)
- DKP 2.8.0 Customer Incidents and Resolved Issues (see page 1959)
- DKP 2.8.0 Deprecations (see page 1960)

13.1.2 Additional resources

- For more information about working with native Kubernetes, see the Kubernetes documentation[1419].
- For a full list of attributed 3rd party software, see http://d2iq.com/legal/3rd .

13.1.3 DKP 2.8.0 Features and Enhancements

The following improvements are included in this release:

13.1.3.1 Kubernetes 1.28.7 Support

Kubernetes 1.28.7 enables you to benefit from the latest features and security fixes in upstream Kubernetes. This release comes with many enhancements[1420]that you can benefit from such as Expanded DNS Configuration[1421]and more.

To read more about major features in this release, visit this page[1422] and Kubernetes 1.28 release[1423].

13.1.3.2 Create an OS Package Bundle for an Air-gapped Environment using Konvoy Image Builder (KIB)

In previous releases, package bundles were downloaded from D2IQ. Now, the air-gapped packages can be built locatlly after downloading the original air-gapped bundle and untarring it. To build the air-gapped package bundle for an Operating System (OS), you would run a new KIB command `create-package-bundle.` In the location where you untar the downloaded package, there is a bundles directory. All the steps to create an OS package bundle[1424] for a particular OS are located there. An example for creating the bundle is to run:

```
./konvoy-image create-package-bundle  --os=centos-7.9  --output-directory=artifacts/
```

This builds an OS bundle for Centos-7.9 using the Kubernetes version defined in the default YAML[1425] - `ansible/group_vars/all/defaults.yaml.`

1419 https://kubernetes.io/docs/home/
1420 https://github.com/orgs/kubernetes/projects/140/views/1
1421 https://github.com/kubernetes/enhancements/issues/2595
1422 https://github.com/kubernetes/sig-release/blob/master/releases/release-1.28/README.md
1423 https://kubernetes.io/blog/2023/08/15/kubernetes-v1-28-release/
1424 https://github.com/mesosphere/konvoy-image-builder/blob/main/docs/cli/konvoy-image_create-package-bundle.md
1425 https://github.com/mesosphere/konvoy-image-builder/blob/main/ansible/group_vars/all/defaults.yaml#L8

13.1.3.3 Konvoy Image Builder (KIB) 2.9.7[1426]

The new release of Konvoy Image Builder:

- Upgrades the defaults Kubernetes version to v1.28.7
- Updates the `containerd` version to 1.6.28

13.1.3.4 DKP Insights

Welcome to DKP Insights 1.1.0!

This release maintains compatibility and support for packages used in Insights.

13.1.3.4.1 Supported Kubernetes Versions

Insights supports the same Kubernetes version as the DKP platform[14271428].

13.1.4 DKP 2.8.0 Supported Kubernetes Versions

13.1.4.1 Deploying Clusters Versions

The latest supported Kubernetes versions for DKP are listed below. In DKP 2.8.1, the Kubernetes version installed by default is 1.28.7. The newest and oldest Kubernetes versions used with DKP must be within one minor version.

- Latest supported Kubernetes version is at **1.28.7 for deploying clusters**
- Other Kubernetes versions are supported at **1.27.9 for deploying clusters in EKS**

Kubernetes 1.28.x enables you to benefit from the latest features and security fixes in upstream Kubernetes. This release comes with 60 enhancements that you can benefit from such as Node log access via Kubernetes API[1429], Seccomp profile defaulting, and much more.

To read more about major features in this release, visit this page[1430] and https://kubernetes.io/blog/2023/04/11/kubernetes-v1-27-release/ .

13.1.4.2 Attaching Clusters Versions

DKP 2.8.1 supports attaching clusters with the following Kubernetes versions:

1426 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.9.7
1427 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/214630401
1428 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/214630401
1429 https://github.com/kubernetes/enhancements/issues/2258
1430 https://github.com/kubernetes/sig-release/blob/master/releases/release-1.27/README.md

Product	Compatible Kubernetes Versions
EKS	1.27.9
AKS	1.28.x
GKE	1.27.x

Attaching Kubernetes clusters with versions greater than n-1 is not recommended.

When attempting to attach a cluster with a lower Kubernetes version than the DKP default, you need to build and use an image with that lower version of Kubernetes. See Konvoy Image Builder (see page 1626) for documentation on building images. Failure to do this could result in an error.

13.1.5 DKP 2.8.0 Kubernetes Major Updates and Deprecations

Before upgrading, we **strongly recommend** verifying your current setup against the information on this page and reading more about Kubernetes' new features in Kubernetes upgrade notes[1431].

13.1.5.1 Deprecated API Services

With this version, Kubernetes stopped serving several API versions for `CSIStorageCapacity` services. For more information on the deprecated API versions for each of these services, refer to https://kubernetes.io/docs/reference/using-api/deprecation-guide/#v1-27.

13.1.5.2 Removed OCI Image Registry

Two releases ago, DKP started using the upstream `registry.k8s.io` registry instead of the previous `k8s.gcr.io` registry for all new and upgraded clusters. Kubernetes has now permanently frozen the images as explained in Freeze k8s.gcr.io image registry[1432].

Ensure you add `registry.k8s.io` to any firewall or proxy allowlists.

1431 https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.27.md
1432 https://kubernetes.io/blog/2023/04/11/kubernetes-v1-27-release/#freeze-k8s-gcr-io-image-registry

13.1.6 DKP 2.8.0 Components and Applications

The following are component and application versions for DKP.

13.1.6.1 Components

Component Name	Version
Cluster API Core (CAPI)	1.5.3-d2iq.0
Cluster API **Amazon AWS** Infrastructure Provider (CAPA)	2.2.4
Cluster API **Google Cloud** Infrastructure Provider (CAPG)	1.5.0
Cluster API **Pre-provisioned** Infrastructure Provider (CAPPP)	0.19.1
Cluster API **vSphere** Infrastructure Provider (CAPV)	1.8.1
Cluster API **Azure** Infrastructure Provider (CAPZ)	1.10.5
Cluster API for **VMware Cloud Director** (CAPVCD)	1.2.0-d2iq.1
Konvoy Image Builder (KIB)	2.9.7[1433]
containerd	1.6.28-d2iq.1
etcd	3.5.10-0
Calico	3.26.3[1434]
Cluster Autoscaler	1.27.3

1433 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.9.7
1434 https://docs.tigera.io/archive

Component Name	Version
CSI_VERSION	Default Storage Providers in DKP (see page 110) See this page for versions and driver table.
Metal LB	0.12.1
Node Feature Discovery	0.14.1

13.1.6.2 Applications

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Ai Navigator App	ai-navigator-app	0.2.2	• chart: 0.2.1 • ai-navigator-app: 0.1.0	Link (see page 1950)	Link[1435]
Ai Navigator Cluster Info Agent	ai-navigator-cluster-info-agent	0.1.1	• chart: 0.1.1 • ai-navigator-cluster-info-agent: 0.1.0	Link (see page 1950)	Link[1436]
Centralized Grafana	centralized-grafana	48.3.2	• chart: 48.3.2[1437] • prometheus-operator: 0.66.0 • grafana: 10.0.3	Link[1438]	Link[1439]
Centralized Kubecost	centralized-kubecost	0.37.3	• chart: 0.37.3[1440] • kubecost: 1.106.7	Link[1441]	Link[1442]

1435 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/ai-navigator-app/0.2.2/defaults/cm.yaml
1436 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/ai-navigator-cluster-info-agent/0.1.1/defaults/cm.yaml
1437 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/48.3.2
1438 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/48.3.2?modal=values
1439 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/centralized-grafana/48.3.2/defaults/cm.yaml
1440 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.37.3
1441 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.37.3?modal=values
1442 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/centralized-kubecost/0.37.3/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Cert Manager	cert-manager	1.14.2	• chart: 1.14.2[1443] • cert-manager: 1.14.2	Link[1444]	Link[1445]
Chartmuseum	chartmuseum	3.10.2	• chart: 3.10.2[1446] • chartmuseum: 3.10.2	Link[1447]	Link[1448]
Dex	dex	2.13.11	• chart: 2.13.11[1449] • dex: 2.37.0	Link[1450]	Link[1451]
Dex K8s Authenticator	dex-k8s-authenticator	1.3.2	• chart: 1.3.2[1452] • dex-k8s-authenticator: 1.3.2	Link[1453]	Link[1454]
Dkp Insights	dkp-insights	1.1.2	• chart: 1.1.2 • dkp-insights: 1.1.2	Link (see page 1950)	Link[1455]
DKP Insights Management	dkp-insights-management	1.1.2	• chart: 1.1.2 • dkp-insights-management: 1.1.2	N/A	Link[1456]

1443 https://artifacthub.io/packages/helm/cert-manager/cert-manager/1.14.2
1444 https://artifacthub.io/packages/helm/cert-manager/cert-manager/1.14.2?modal=values
1445 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/cert-manager/1.14.2/defaults/cm.yaml
1446 https://artifacthub.io/packages/helm/chartmuseum/chartmuseum/3.10.2
1447 https://artifacthub.io/packages/helm/chartmuseum/chartmuseum/3.10.2?modal=values
1448 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/chartmuseum/3.10.2/defaults/cm.yaml
1449 https://artifacthub.io/packages/helm/mesosphere-stable/dex/2.13.11
1450 https://artifacthub.io/packages/helm/mesosphere-stable/dex/2.13.11?modal=values
1451 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/dex/2.13.11/defaults/cm.yaml
1452 https://artifacthub.io/packages/helm/mesosphere/dex-k8s-authenticator/1.3.2
1453 https://artifacthub.io/packages/helm/mesosphere/dex-k8s-authenticator/1.3.2?modal=values
1454 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/dex-k8s-authenticator/1.3.2/defaults/cm.yaml
1455 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/dkp-insights/1.1.2/defaults/cm.yaml
1456 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/dkp-insights-management/1.1.2/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
External DNS	external-dns	7.1.2	• chart: 7.1.2[1457] • external-dns: 0.14.1	Link[1458]	Link[1459]
Fluent Bit	fluent-bit	0.43.0	• chart: 0.43.0[1460] • fluent-bit: 2.2.2	Link[1461]	Link[1462]
Gatekeeper	gatekeeper	3.14.1	• chart: 3.14.1[1463] • gatekeeper: 3.14.1	Link[1464]	Link[1465]
Gitea	gitea	8.2.1	• chart: 8.2.0[1466] • gitea: 1.19.2	Link[1467]	Link[1468]
Grafana Logging	grafana-logging	6.60.2	• chart: 6.60.1[1469] • grafana: 10.1.2	Link[1470]	Link[1471]
Grafana Loki	grafana-loki	0.78.3	• chart: 0.78.3[1472] • loki: 2.9.4	Link[1473]	Link[1474]

1457 https://artifacthub.io/packages/helm/bitnami/external-dns/7.1.2
1458 https://artifacthub.io/packages/helm/bitnami/external-dns/7.1.2?modal=values
1459 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/external-dns/7.1.2/defaults/cm.yaml
1460 https://artifacthub.io/packages/helm/fluent/fluent-bit/0.43.0
1461 https://artifacthub.io/packages/helm/fluent/fluent-bit/0.43.0?modal=values
1462 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/fluent-bit/0.43.0/defaults/cm.yaml
1463 https://artifacthub.io/packages/helm/gatekeeper/gatekeeper/3.14.1
1464 https://artifacthub.io/packages/helm/gatekeeper/gatekeeper/3.14.1?modal=values
1465 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/gatekeeper/3.14.1/defaults/cm.yaml
1466 https://artifacthub.io/packages/helm/gitea/gitea/8.2.0
1467 https://artifacthub.io/packages/helm/gitea/gitea/8.2.0?modal=values
1468 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/gitea/8.2.1/defaults/cm.yaml
1469 https://artifacthub.io/packages/helm/grafana/grafana/6.60.1
1470 https://artifacthub.io/packages/helm/grafana/grafana/6.60.1?modal=values
1471 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/grafana-logging/6.60.2/defaults/cm.yaml
1472 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.78.3
1473 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.78.3?modal=values
1474 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/grafana-loki/0.78.3/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Istio	istio	1.20.2	• chart: 1.20.2[1475] • istio: 1.20.2	Link[1476]	Link[1477]
Jaeger	jaeger	2.50.1	• chart: 2.50.1[1478] • jaeger: 1.52.0	Link[1479]	Link[1480]
Karma	karma	2.0.4	• chart: 2.0.2[1481] • karma: 0.70	Link[1482]	Link[1483]
Kiali	kiali	1.79.0	• chart: 1.79.0[1484] • kiali: 1.79.0	Link[1485]	Link[1486]
Knative	knative	1.10.5	• chart: 1.10.4[1487] • knative: 1.10.0	Link[1488]	Link[1489]
Flux	kommander-flux	2.2.3	• chart: N/A • flux: 2.2.3	N/A	N/A
Kommander Ui	kommander-ui	11.4.5	• chart: 11.4.5 • kommander-ui: 11.4.5	Link (see page 1950)	Link[1490]

1475 https://artifacthub.io/packages/helm/mesosphere/istio/1.20.2
1476 https://artifacthub.io/packages/helm/mesosphere/istio/1.20.2?modal=values
1477 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/istio/1.20.2/defaults/cm.yaml
1478 https://artifacthub.io/packages/helm/jaegertracing/jaeger-operator/2.50.1
1479 https://artifacthub.io/packages/helm/jaegertracing/jaeger-operator/2.50.1?modal=values
1480 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/jaeger/2.50.1/defaults/cm.yaml
1481 https://artifacthub.io/packages/helm/mesosphere-stable/karma/2.0.2
1482 https://artifacthub.io/packages/helm/mesosphere-stable/karma/2.0.2?modal=values
1483 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/karma/2.0.4/defaults/cm.yaml
1484 https://artifacthub.io/packages/helm/kiali/kiali-operator/1.79.0
1485 https://artifacthub.io/packages/helm/kiali/kiali-operator/1.79.0?modal=values
1486 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/kiali/1.79.0/defaults/cm.yaml
1487 https://artifacthub.io/packages/helm/mesosphere/knative/1.10.4
1488 https://artifacthub.io/packages/helm/mesosphere/knative/1.10.4?modal=values
1489 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/knative/1.10.5/defaults/cm.yaml
1490 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/kommander-ui/11.4.5/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Kube OIDC Proxy	kube-oidc-proxy	0.3.4	• chart: 0.3.4[1491] • kube-oidc-proxy: 0.3.0	Link[1492]	Link[1493]
Kube Prometheus Stack	kube-prometheus-stack	48.3.2	• chart: 48.3.2[1494] • prometheus-operator: 0.66.0 • grafana: 10.0.3 • prometheus: 2.45.0 • prometheus-alertmanager: 0.25.0	Link[1495]	Link[1496]
Kubecost	kubecost	0.37.3	• chart: 0.37.3[1497] • kubecost: 1.106.7	Link[1498]	Link[1499]
Kubefed	kubefed	0.10.5	• chart: 0.10.4[1500] • kubefed: 0.10.5	Link[1501]	Link[1502]
Kubernetes Dashboard	kubernetes-dashboard	6.0.9	• chart: 6.0.8[1503] • kubernetes-dashboard: 2.7.0	Link[1504]	Link[1505]

1491 https://artifacthub.io/packages/helm/mesosphere/kube-oidc-proxy/0.3.4
1492 https://artifacthub.io/packages/helm/mesosphere/kube-oidc-proxy/0.3.4?modal=values
1493 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/kube-oidc-proxy/0.3.4/defaults/cm.yaml
1494 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/48.3.2
1495 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/48.3.2?modal=values
1496 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/kube-prometheus-stack/48.3.2/defaults/cm.yaml
1497 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.37.3
1498 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.37.3?modal=values
1499 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/kubecost/0.37.3/defaults/cm.yaml
1500 https://artifacthub.io/packages/helm/kubefed/kubefed/0.10.4
1501 https://artifacthub.io/packages/helm/kubefed/kubefed/0.10.4?modal=values
1502 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/kubefed/0.10.5/defaults/cm.yaml
1503 https://artifacthub.io/packages/helm/k8s-dashboard/kubernetes-dashboard/6.0.8
1504 https://artifacthub.io/packages/helm/k8s-dashboard/kubernetes-dashboard/6.0.8?modal=values
1505 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/kubernetes-dashboard/6.0.9/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Kubetunnel	kubetunnel	0.0.32	• chart: 0.0.31 • kubetunnel: 0.0.31	N/A	Link[1506]
Logging Operator	logging-operator	4.2.4	• chart: 4.2.3[1507] • logging-operator: 4.2.2 • logging-operator-logging: 4.2.3	Link[1508]	Link[1509]
NFS Server Provisioner	nfs-server-provisioner	0.6.1	• chart: 0.6.1[1510] • nfs-server-provisioner: 2.3.0	Link[1511]	Link[1512]
NVIDIA GPU Operator	nvidia-gpu-operator	23.9.2	• chart: 23.9.2[1513] • nvidia-gpu-operator: 23.9.2	Link[1514]	Link[1515]
Grafana (project)	project-grafana-logging	6.60.2	• chart: 6.60.1[1516] • grafana: 10.1.2	Link[1517]	Link[1518]

1506 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/kubetunnel/0.0.32/defaults/cm.yaml
1507 https://artifacthub.io/packages/helm/banzaicloud-stable/logging-operator/4.2.3
1508 https://artifacthub.io/packages/helm/banzaicloud-stable/logging-operator/4.2.3?modal=values
1509 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/logging-operator/4.2.4/defaults/cm.yaml
1510 https://artifacthub.io/packages/helm/mesosphere/nfs-server-provisioner/0.6.1
1511 https://artifacthub.io/packages/helm/mesosphere/nfs-server-provisioner/0.6.1?modal=values
1512 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/nfs-server-provisioner/0.6.1/defaults/cm.yaml
1513 https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/getting-started.html#chart-customization-options
1514 https://raw.githubusercontent.com/NVIDIA/gpu-operator/v23.9.2/v%25s/deployments/gpu-operator/values.yaml
1515 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/nvidia-gpu-operator/23.9.2/defaults/cm.yaml
1516 https://artifacthub.io/packages/helm/grafana/grafana/6.60.1
1517 https://artifacthub.io/packages/helm/grafana/grafana/6.60.1?modal=values
1518 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/project-grafana-logging/6.60.2/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Grafana Loki (project)	project-grafana-loki	0.78.3	• chart: 0.78.3[1519] • loki: 2.9.4	Link[1520]	Link[1521]
Prometheus Adapter	prometheus-adapter	4.9.0	• chart: 4.9.0[1522] • prometheus-adapter: 0.11.2	Link[1523]	Link[1524]
Reloader	reloader	1.0.65	• chart: 1.0.65[1525] • reloader: 1.0.65	Link[1526]	Link[1527]
Rook Ceph	rook-ceph	1.12.7	• chart: 1.12.6[1528] • rook-ceph: 1.12.6	Link[1529]	Link[1530]
Rook Ceph Cluster	rook-ceph-cluster	1.12.7	• chart: 1.12.6[1531] • rook-ceph: 1.12.6 • rook-ceph-cluster: 17.2.6	Link[1532]	Link[1533]

1519 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.78.3
1520 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.78.3?modal=values
1521 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/project-grafana-loki/0.78.3/defaults/cm.yaml
1522 https://artifacthub.io/packages/helm/prometheus-community/prometheus-adapter/4.9.0
1523 https://artifacthub.io/packages/helm/prometheus-community/prometheus-adapter/4.9.0?modal=values
1524 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/prometheus-adapter/4.9.0/defaults/cm.yaml
1525 https://github.com/stakater/Reloader/tree/v1.0.65/README.md#helm-charts
1526 https://artifacthub.io/packages/helm/stakater/reloader/1.0.65?modal=values
1527 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/reloader/1.0.65/defaults/cm.yaml
1528 https://github.com/rook/rook/tree/vv1.12.6/Documentation/Helm-Charts/operator-chart.md
1529 https://raw.githubusercontent.com/rook/rook/v1.12.6/v%25s/deploy/charts/rook-ceph/values.yaml
1530 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/rook-ceph/1.12.7/defaults/cm.yaml
1531 https://github.com/rook/rook/tree/vv1.12.6/Documentation/Helm-Charts/ceph-cluster-chart.md
1532 https://raw.githubusercontent.com/rook/rook/v1.12.6/v%25s/deploy/charts/rook-ceph-cluster/values.yaml
1533 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/rook-ceph-cluster/1.12.7/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Thanos	thanos	13.2.1	• chart: 13.2.1[1534] • thanos: 0.34.0	Link[1535]	Link[1536]
Traefik	traefik	26.0.0	• chart: 26.0.0[1537] • traefik: 2.10.6	Link[1538]	Link[1539]
Traefik ForwardAuth	traefik-forward-auth	0.3.11	• chart: 0.3.10[1540] • traefik-forward-auth: 3.1.0	Link[1541]	Link[1542]
Velero	velero	5.2.2	• chart: 5.2.2[1543] • velero: 1.12.3	Link[1544]	Link[1545]

13.1.7 DKP 2.8.0 Known Issues and Limitations

The following items are known issues with this release.

13.1.7.1 AWS Custom AMI Required for Kubernetes Version

Previous versions of DKP would default to using upstream AMIs published by the CAPA (Cluster API AWS) project when building AWS clusters if you did not specify your own AMI. However, those images are not currently available for the Kubernetes version used in the 2.8.0 release.

As a result, starting with this release of DKP, the behavior of the DKP `create cluster aws` command has been changed. It no longer defaults to using the upstream AMIs and instead requires that you specify an

1534 https://artifacthub.io/packages/helm/bitnami/thanos/13.2.1
1535 https://artifacthub.io/packages/helm/bitnami/thanos/13.2.1?modal=values
1536 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/thanos/13.2.1/defaults/cm.yaml
1537 https://artifacthub.io/packages/helm/traefik/traefik/26.0.0
1538 https://artifacthub.io/packages/helm/traefik/traefik/26.0.0?modal=values
1539 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/traefik/26.0.0/defaults/cm.yaml
1540 https://artifacthub.io/packages/helm/mesosphere/traefik-forward-auth/0.3.10
1541 https://artifacthub.io/packages/helm/mesosphere/traefik-forward-auth/0.3.10?modal=values
1542 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/traefik-forward-auth/0.3.11/defaults/cm.yaml
1543 https://artifacthub.io/packages/helm/vmware-tanzu/velero/5.2.2
1544 https://artifacthub.io/packages/helm/vmware-tanzu/velero/5.2.2?modal=values
1545 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.0/services/velero/5.2.2/defaults/cm.yaml

AMI built using Konvoy Image Builder (KIB) (see page 1626), or by explicitly requesting that it use the upstream images.

For more information on using a custom AMI in cluster creation or during the upgrade process, refer to these topics:

- AWS: Create the Management Cluster (see page 300)
- Enterprise: Upgrade the Management Cluster Kubernetes Version (see page 1735)
- Essential Upgrade Kubernetes Version (see page 1753)

13.1.7.2 vSphere and Ubuntu using Konvoy Image Builder Issue

Building a Konvoy Image Builder (KIB) image on vSphere using Ubuntu 20.04 with `cloud-init 23.3.3-0ubuntu0~20.04.1` will fail in the 2.8.0 version of DKP. The issue will be resolved in an upcoming KIB patch release.

13.1.7.3 Known CVE's

Since the release of DKP 2.6, DKP has been scanning Catalog applications for CVEs. Beginning with DKP 2.7, only the latest version of each Catalog application will be scanned (and have its critical CVEs mitigated). ` For more information about the known CVE's for compatible catalog apps, refer to D2iQ Security Updates (see page 1992).

Custom Banner on Cluster will not Save on Upgrade from DKP 2.7.0 to 2.8.0, if you have a custom banner it will no longer be available in the UI. You can then create a new banner.

13.1.8 DKP 2.8.0 Customer Incidents and Resolved Issues

The following resolved incidents are fixed or corrected in this release.

13.1.8.1 Improved Error Message when Pre-provisioned KIB Jobs Fail

Previously, during the installation of a pre-provisioned cluster, it could be difficult to determine why provisioning of the cluster nodes failed. The error handling of the Konvoy Image Builder (KIB) jobs that provision the nodes has been improved to aid in troubleshooting.

D2IQ-99310

13.1.8.2 Dex GRPC Endpoint Disabled

Previously, DKP enabled the GRPC endpoint on Dex but it was not used by any other components in the system. The endpoint is now disabled by default.

D2IQ-99912

13.1.8.3 Dex DA Certificate Expiration Causes the UI to be Unreachable

A misconfiguration of the Dex certificate authority (CA) issuer and the `dex-client-tls` certificate resulted in the `dex-client-tls` certificate being invalid after approximately 90 days.
When the certificate becomes invalid, the DKP UI is no longer accessible and displays an “Internal Server Error - Failed to Retrieve Connector List” message.

The problem has been corrected and a helpful error message added. In the meantime, if you experience this issue, contact support for a workaround.

D2IQ-100649

13.1.8.3.1 CAPA AMIs are Not Available for the Kubernetes Version used in DKP 2.8.0

AWS AMIs for some Kubernetes versions may not be available. We recommend using Konvoy Image Builder[1546] (KIB) to create a custom AMI.

- Do not use the upstream values as defaults for the `--ami-*` flags.
- If you specify any of these flags for AMI ID lookup, specify all of them: `--ami-base-os`, `--ami-format`, and `--ami-owner`.

See Also: DKP 2.8.0 Known Issues and Limitations (see page 1958)

D2IQ-100175

13.1.9 DKP 2.8.0 Deprecations

13.1.9.1 OS Deprecations

In the next release after 2.8.1, DKP will drop support for the following Operating Systems (OS):

- RHEL 7.9
- RHEL 8.4
- CentOS 7.9

For RHEL users, the recommendation is to upgrade to a supported version of RHEL 8.6 or RHEL 8.8.

For CentOS users, you can choose to migrate to any supported operating system in our support matrix found on the Supported Infrastructure Operating Systems (see page 67) page. We support Rocky Linux, RHEL and Ubuntu.

13.2 DKP 2.8.1 Release Notes

DKP® version 2.8.1 was released on June 6, 2024.

1546 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/369393668

Download DKP

You must be a registered user and logged on to the support portal to download this product. New customers must contact their sales representative or sales@d2iq.com[1547] before attempting to download or install DKP.

13.2.1 Release Summary

Welcome to D2iQ Kubernetes Platform (DKP) 2.8.1! This release focuses on improved features for multi-tenancy for scalable and resilient fleet management. Below, you will find information about the following:

13.2.2 Additional resources

- For more information about working with native Kubernetes, see the Kubernetes documentation[1548].
- For a full list of attributed 3rd party software, see http://d2iq.com/legal/3rd .

13.2.3 DKP 2.8.1 Features and Enhancements

The following improvements are included in this release:

13.2.3.1 Kubernetes 1.28.7 Support

Kubernetes 1.28.7 enables you to benefit from the latest features and security fixes in upstream Kubernetes. This release comes with many enhancements[1549]that you can benefit from such as Expanded DNS Configuration[1550]and more.

To read more about major features in this release, visit this page[1551] and Kubernetes 1.28 release[1552].

1547 mailto:sales@d2iq.com
1548 https://kubernetes.io/docs/home/
1549 https://github.com/orgs/kubernetes/projects/140/views/1
1550 https://github.com/kubernetes/enhancements/issues/2595
1551 https://github.com/kubernetes/sig-release/blob/master/releases/release-1.28/README.md
1552 https://kubernetes.io/blog/2023/08/15/kubernetes-v1-28-release/

13.2.3.2 Create an OS Package Bundle for an Air-gapped Environment using Konvoy Image Builder (KIB)

In previous releases, package bundles were downloaded from D2IQ. Now, the air-gapped packages can be built locatlly after downloading the original air-gapped bundle and untarring it. To build the air-gapped package bundle for an Operating System (OS), you would run a new KIB command `create-package-bundle.` In the location where you untar the downloaded package, there is a bundles directory. All the steps to create an OS package bundle[1553] for a particular OS are located there. An example for creating the bundle is to run:

```
./konvoy-image create-package-bundle  --os=centos-7.9  --output-directory=artifacts/
```

This builds an OS bundle for Centos-7.9 using the Kubernetes version defined in the default YAML[1554] - `ansible/group_vars/all/defaults.yaml.`

13.2.3.3 Konvoy Image Builder (KIB) 2.9.7[1555]

The new release of Konvoy Image Builder:

- Upgrades the defaults Kubernetes version to v1.28.7
- Updates the `containerd` version to 1.6.28

13.2.3.4 DKP Insights

Welcome to DKP Insights 1.1.0!

This release maintains compatibility and support for packages used in Insights.

13.2.3.4.1 Supported Kubernetes Versions

Insights supports the same Kubernetes version as the DKP platform[15561557].

1553 https://github.com/mesosphere/konvoy-image-builder/blob/main/docs/cli/konvoy-image_create-package-bundle.md
1554 https://github.com/mesosphere/konvoy-image-builder/blob/main/ansible/group_vars/all/defaults.yaml#L8
1555 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.9.7
1556 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/214630401
1557 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/214630401

13.2.4 DKP 2.8.1 Supported Kubernetes Versions

13.2.4.1 Deploying Clusters Versions

The latest supported Kubernetes versions for DKP are listed below. In DKP 2.8.1, the Kubernetes version installed by default is 1.28.7. The newest and oldest Kubernetes versions used with DKP must be within one minor version.

- Latest supported Kubernetes version is at **1.28.7 for deploying clusters**
- Other Kubernetes versions are supported at **1.27.9 for deploying clusters in EKS**

Kubernetes 1.28.x enables you to benefit from the latest features and security fixes in upstream Kubernetes. This release comes with 60 enhancements that you can benefit from such as Node log access via Kubernetes API[1558], Seccomp profile defaulting, and much more.

To read more about major features in this release, visit this page[1559] and https://kubernetes.io/blog/2023/04/11/kubernetes-v1-27-release/ .

13.2.4.2 Attaching Clusters Versions

DKP 2.8.1 supports attaching clusters with the following Kubernetes versions:

Product	Compatible Kubernetes Versions
EKS	1.27.9
AKS	1.28.x
GKE	1.27.x

Attaching Kubernetes clusters with versions greater than n-1 is not recommended.

When attempting to attach a cluster with a lower Kubernetes version than the DKP default, you need to build and use an image with that lower version of Kubernetes. See Konvoy Image Builder (see page 1626) for documentation on building images. Failure to do this could result in an error.

1558 https://github.com/kubernetes/enhancements/issues/2258
1559 https://github.com/kubernetes/sig-release/blob/master/releases/release-1.27/README.md

13.2.5 DKP 2.8.1 Kubernetes Major Updates and Deprecations

Before upgrading, we **strongly recommend** verifying your current setup against the information on this page and reading more about Kubernetes' new features in Kubernetes upgrade notes[1560].

13.2.5.1 Deprecated API Services

With this version, Kubernetes stopped serving several API versions for `CSIStorageCapacity` services. For more information on the deprecated API versions for each of these services, refer to https://kubernetes.io/docs/reference/using-api/deprecation-guide/#v1-27.

13.2.5.2 Removed OCI Image Registry

Two releases ago, DKP started using the upstream `registry.k8s.io` registry instead of the previous `k8s.gcr.io` registry for all new and upgraded clusters. Kubernetes has now permanently frozen the images as explained in Freeze k8s.gcr.io image registry[1561].

Ensure you add `registry.k8s.io` to any firewall or proxy allowlists.

13.2.6 DKP 2.8.1 Components and Applications

The following are component and application versions for DKP.

13.2.6.1 Components

Component Name	Version
Cluster API Core (CAPI)	1.5.3-d2iq.0
Cluster API **Amazon AWS** Infrastructure Provider (CAPA)	2.2.4
Cluster API **Google Cloud** Infrastructure Provider (CAPG)	1.5.0
Cluster API **Pre-provisioned** Infrastructure Provider (CAPPP)	0.19.1

1560 https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.27.md
1561 https://kubernetes.io/blog/2023/04/11/kubernetes-v1-27-release/#freeze-k8s-gcr-io-image-registry

Component Name	Version
Cluster API **vSphere** Infrastructure Provider (CAPV)	1.8.1
Cluster API **Azure** Infrastructure Provider (CAPZ)	1.10.5
Cluster API for **VMware Cloud Director** (CAPVCD)	1.2.0-d2iq.1
Konvoy Image Builder (KIB)	2.9.7[1562]
containerd	1.6.28-d2iq.1
etcd	3.5.10-0
Calico	3.26.3[1563]
Cluster Autoscaler	1.27.3
CSI_VERSION	Default Storage Providers in DKP (see page 110) See this page for versions and driver table.
Metal LB	0.12.1
Node Feature Discovery	0.14.1

1562 https://github.com/mesosphere/konvoy-image-builder/releases/tag/v2.9.7
1563 https://docs.tigera.io/archive

13.2.6.2 Applications

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Ai Navigator App	ai-navigator-app	0.2.2	• chart: 0.2.1 • ai-navigator-app: 0.1.0	Link	Link[1564]
Ai Navigator Cluster Info Agent	ai-navigator-cluster-info-agent	0.1.1	• chart: 0.1.1 • ai-navigator-cluster-info-agent: 0.1.0	Link	Link[1565]
Centralized Grafana	centralized-grafana	48.3.2	• chart: 48.3.2[1566] • prometheus-operator: 0.66.0 • grafana: 10.0.3	Link[1567]	Link[1568]
Centralized Kubecost	centralized-kubecost	0.37.3	• chart: 0.37.3[1569] • kubecost: 1.106.7	Link[1570]	Link[1571]
Cert Manager	cert-manager	1.14.2	• chart: 1.14.2[1572] • cert-manager: 1.14.2	Link[1573]	Link[1574]

1564 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/ai-navigator-app/0.2.2/defaults/cm.yaml
1565 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/ai-navigator-cluster-info-agent/0.1.1/defaults/cm.yaml
1566 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/48.3.2
1567 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/48.3.2?modal=values
1568 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/centralized-grafana/48.3.2/defaults/cm.yaml
1569 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.37.3
1570 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.37.3?modal=values
1571 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/centralized-kubecost/0.37.3/defaults/cm.yaml
1572 https://artifacthub.io/packages/helm/cert-manager/cert-manager/1.14.2
1573 https://artifacthub.io/packages/helm/cert-manager/cert-manager/1.14.2?modal=values
1574 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/cert-manager/1.14.2/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Chartmuseum	chartmuseum	3.10.2	• chart: 3.10.2[1575] • chartmuseum: 3.10.2	Link[1576]	Link[1577]
Dex	dex	2.13.11	• chart: 2.13.11[1578] • dex: 2.37.0	Link[1579]	Link[1580]
Dex K8s Authenticator	dex-k8s-authenticator	1.3.2	• chart: 1.3.2[1581] • dex-k8s-authenticator: 1.3.2	Link[1582]	Link[1583]
Dkp Insights	dkp-insights	1.1.2	• chart: 1.1.2 • dkp-insights: 1.1.2	Link	Link[1584]
DKP Insights Management	dkp-insights-management	1.1.2	• chart: 1.1.2 • dkp-insights-management: 1.1.2	N/A	Link[1585]
External DNS	external-dns	7.1.2	• chart: 7.1.2[1586] • external-dns: 0.14.1	Link[1587]	Link[1588]

1575 https://artifacthub.io/packages/helm/chartmuseum/chartmuseum/3.10.2
1576 https://artifacthub.io/packages/helm/chartmuseum/chartmuseum/3.10.2?modal=values
1577 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/chartmuseum/3.10.2/defaults/cm.yaml
1578 https://artifacthub.io/packages/helm/mesosphere-stable/dex/2.13.11
1579 https://artifacthub.io/packages/helm/mesosphere-stable/dex/2.13.11?modal=values
1580 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/dex/2.13.11/defaults/cm.yaml
1581 https://artifacthub.io/packages/helm/mesosphere/dex-k8s-authenticator/1.3.2
1582 https://artifacthub.io/packages/helm/mesosphere/dex-k8s-authenticator/1.3.2?modal=values
1583 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/dex-k8s-authenticator/1.3.2/defaults/cm.yaml
1584 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/dkp-insights/1.1.2/defaults/cm.yaml
1585 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/dkp-insights-management/1.1.2/defaults/cm.yaml
1586 https://artifacthub.io/packages/helm/bitnami/external-dns/7.1.2
1587 https://artifacthub.io/packages/helm/bitnami/external-dns/7.1.2?modal=values
1588 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/external-dns/7.1.2/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Fluent Bit	fluent-bit	0.43.1	• chart: 0.43.1[1589] • fluent-bit: 2.2.2	Link[1590]	Link[1591]
Gatekeeper	gatekeeper	3.14.1	• chart: 3.14.1[1592] • gatekeeper: 3.14.1	Link[1593]	Link[1594]
Gitea	gitea	8.2.1	• chart: 8.2.0[1595] • gitea: 1.19.2	Link[1596]	Link[1597]
Grafana Logging	grafana-logging	6.60.2	• chart: 6.60.1[1598] • grafana: 10.1.2	Link[1599]	Link[1600]
Grafana Loki	grafana-loki	0.78.3	• chart: 0.78.3[1601] • loki: 2.9.4	Link[1602]	Link[1603]
Istio	istio	1.20.2	• chart: 1.20.2[1604] • istio: 1.20.2	Link[1605]	Link[1606]

1589 https://artifacthub.io/packages/helm/fluent/fluent-bit/0.43.1
1590 https://artifacthub.io/packages/helm/fluent/fluent-bit/0.43.1?modal=values
1591 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/fluent-bit/0.43.1/defaults/cm.yaml
1592 https://artifacthub.io/packages/helm/gatekeeper/gatekeeper/3.14.1
1593 https://artifacthub.io/packages/helm/gatekeeper/gatekeeper/3.14.1?modal=values
1594 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/gatekeeper/3.14.1/defaults/cm.yaml
1595 https://artifacthub.io/packages/helm/gitea/gitea/8.2.0
1596 https://artifacthub.io/packages/helm/gitea/gitea/8.2.0?modal=values
1597 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/gitea/8.2.1/defaults/cm.yaml
1598 https://artifacthub.io/packages/helm/grafana/grafana/6.60.1
1599 https://artifacthub.io/packages/helm/grafana/grafana/6.60.1?modal=values
1600 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/grafana-logging/6.60.2/defaults/cm.yaml
1601 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.78.3
1602 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.78.3?modal=values
1603 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/grafana-loki/0.78.3/defaults/cm.yaml
1604 https://artifacthub.io/packages/helm/mesosphere/istio/1.20.2
1605 https://artifacthub.io/packages/helm/mesosphere/istio/1.20.2?modal=values
1606 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/istio/1.20.2/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Jaeger	jaeger	2.50.1	• chart: 2.50.1[1607] • jaeger: 1.52.0	Link[1608]	Link[1609]
Karma	karma	2.0.4	• chart: 2.0.2[1610] • karma: 0.70	Link[1611]	Link[1612]
Kiali	kiali	1.79.0	• chart: 1.79.0[1613] • kiali: 1.79.0	Link[1614]	Link[1615]
Knative	knative	1.10.5	• chart: 1.10.4[1616] • knative: 1.10.0	Link[1617]	Link[1618]
Flux	kommander-flux	2.2.3	• chart: N/A • flux: 2.2.3	N/A	N/A
Kommander Ui	kommander-ui	11.4.6	• chart: 11.4.6 • kommander-ui: 11.4.6	Link	Link[1619]
Kube OIDC Proxy	kube-oidc-proxy	0.3.4	• chart: 0.3.4[1620] • kube-oidc-proxy: 0.3.0	Link[1621]	Link[1622]

1607 https://artifacthub.io/packages/helm/jaegertracing/jaeger-operator/2.50.1
1608 https://artifacthub.io/packages/helm/jaegertracing/jaeger-operator/2.50.1?modal=values
1609 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/jaeger/2.50.1/defaults/cm.yaml
1610 https://artifacthub.io/packages/helm/mesosphere-stable/karma/2.0.2
1611 https://artifacthub.io/packages/helm/mesosphere-stable/karma/2.0.2?modal=values
1612 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/karma/2.0.4/defaults/cm.yaml
1613 https://artifacthub.io/packages/helm/kiali/kiali-operator/1.79.0
1614 https://artifacthub.io/packages/helm/kiali/kiali-operator/1.79.0?modal=values
1615 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/kiali/1.79.0/defaults/cm.yaml
1616 https://artifacthub.io/packages/helm/mesosphere/knative/1.10.4
1617 https://artifacthub.io/packages/helm/mesosphere/knative/1.10.4?modal=values
1618 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/knative/1.10.5/defaults/cm.yaml
1619 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/kommander-ui/11.4.6/defaults/cm.yaml
1620 https://artifacthub.io/packages/helm/mesosphere/kube-oidc-proxy/0.3.4
1621 https://artifacthub.io/packages/helm/mesosphere/kube-oidc-proxy/0.3.4?modal=values
1622 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/kube-oidc-proxy/0.3.4/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Kube Prometheus Stack	kube-prometheus-stack	48.3.2	• chart: 48.3.2[1623] • prometheus-operator: 0.66.0 • grafana: 10.0.3 • prometheus: 2.45.0 • prometheus-alertmanager: 0.25.0	Link[1624]	Link[1625]
Kubecost	kubecost	0.37.3	• chart: 0.37.3[1626] • kubecost: 1.106.7	Link[1627]	Link[1628]
Kubefed	kubefed	0.10.5	• chart: 0.10.4[1629] • kubefed: 0.10.5	Link[1630]	Link[1631]
Kubernetes Dashboard	kubernetes-dashboard	6.0.9	• chart: 6.0.8[1632] • kubernetes-dashboard: 2.7.0	Link[1633]	Link[1634]
Kubetunnel	kubetunnel	0.0.32	• chart: 0.0.31 • kubetunnel: 0.0.31	N/A	Link[1635]

1623 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/48.3.2
1624 https://artifacthub.io/packages/helm/mesosphere/kube-prometheus-stack/48.3.2?modal=values
1625 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/kube-prometheus-stack/48.3.2/defaults/cm.yaml
1626 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.37.3
1627 https://artifacthub.io/packages/helm/mesosphere-stable/kubecost/0.37.3?modal=values
1628 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/kubecost/0.37.3/defaults/cm.yaml
1629 https://artifacthub.io/packages/helm/kubefed/kubefed/0.10.4
1630 https://artifacthub.io/packages/helm/kubefed/kubefed/0.10.4?modal=values
1631 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/kubefed/0.10.5/defaults/cm.yaml
1632 https://artifacthub.io/packages/helm/k8s-dashboard/kubernetes-dashboard/6.0.8
1633 https://artifacthub.io/packages/helm/k8s-dashboard/kubernetes-dashboard/6.0.8?modal=values
1634 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/kubernetes-dashboard/6.0.9/defaults/cm.yaml
1635 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/kubetunnel/0.0.32/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Logging Operator	logging-operator	4.2.5	• chart: 4.2.3[1636] • logging-operator: 4.2.2 • logging-operator-logging: 4.2.3	Link[1637]	Link[1638]
NFS Server Provisioner	nfs-server-provisioner	0.6.1	• chart: 0.6.1[1639] • nfs-server-provisioner: 2.3.0	Link[1640]	Link[1641]
NVIDIA GPU Operator	nvidia-gpu-operator	23.9.2	• chart: 23.9.2[1642] • nvidia-gpu-operator: 23.9.2	Link[1643]	Link[1644]
Grafana (project)	project-grafana-logging	6.60.2	• chart: 6.60.1[1645] • grafana: 10.1.2	Link[1646]	Link[1647]
Grafana Loki (project)	project-grafana-loki	0.78.3	• chart: 0.78.3[1648] • loki: 2.9.4	Link[1649]	Link[1650]

1636 https://artifacthub.io/packages/helm/banzaicloud-stable/logging-operator/4.2.3
1637 https://artifacthub.io/packages/helm/banzaicloud-stable/logging-operator/4.2.3?modal=values
1638 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/logging-operator/4.2.5/defaults/cm.yaml
1639 https://artifacthub.io/packages/helm/mesosphere/nfs-server-provisioner/0.6.1
1640 https://artifacthub.io/packages/helm/mesosphere/nfs-server-provisioner/0.6.1?modal=values
1641 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/nfs-server-provisioner/0.6.1/defaults/cm.yaml
1642 https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/getting-started.html#chart-customization-options
1643 https://raw.githubusercontent.com/NVIDIA/gpu-operator/v23.9.2/v%25s/deployments/gpu-operator/values.yaml
1644 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/nvidia-gpu-operator/23.9.2/defaults/cm.yaml
1645 https://artifacthub.io/packages/helm/grafana/grafana/6.60.1
1646 https://artifacthub.io/packages/helm/grafana/grafana/6.60.1?modal=values
1647 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/project-grafana-logging/6.60.2/defaults/cm.yaml
1648 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.78.3
1649 https://artifacthub.io/packages/helm/grafana/loki-distributed/0.78.3?modal=values
1650 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/project-grafana-loki/0.78.3/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Prometheus Adapter	prometheus-adapter	4.9.0	• chart: 4.9.0[1651] • prometheus-adapter: 0.11.2	Link[1652]	Link[1653]
Reloader	reloader	1.0.65	• chart: 1.0.65[1654] • reloader: 1.0.65	Link[1655]	Link[1656]
Rook Ceph	rook-ceph	1.12.7	• chart: 1.12.6[1657] • rook-ceph: 1.12.6	Link[1658]	Link[1659]
Rook Ceph Cluster	rook-ceph-cluster	1.12.7	• chart: 1.12.6[1660] • rook-ceph: 1.12.6 • rook-ceph-cluster: 17.2.6	Link[1661]	Link[1662]
Thanos	thanos	13.2.1	• chart: 13.2.1[1663] • thanos: 0.34.0	Link[1664]	Link[1665]

1651 https://artifacthub.io/packages/helm/prometheus-community/prometheus-adapter/4.9.0
1652 https://artifacthub.io/packages/helm/prometheus-community/prometheus-adapter/4.9.0?modal=values
1653 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/prometheus-adapter/4.9.0/defaults/cm.yaml
1654 https://github.com/stakater/Reloader/tree/v1.0.65/README.md#helm-charts
1655 https://artifacthub.io/packages/helm/stakater/reloader/1.0.65?modal=values
1656 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/reloader/1.0.65/defaults/cm.yaml
1657 https://github.com/rook/rook/tree/vv1.12.6/Documentation/Helm-Charts/operator-chart.md
1658 https://raw.githubusercontent.com/rook/rook/v1.12.6/v%25s/deploy/charts/rook-ceph/values.yaml
1659 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/rook-ceph/1.12.7/defaults/cm.yaml
1660 https://github.com/rook/rook/tree/vv1.12.6/Documentation/Helm-Charts/ceph-cluster-chart.md
1661 https://raw.githubusercontent.com/rook/rook/v1.12.6/v%25s/deploy/charts/rook-ceph-cluster/values.yaml
1662 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/rook-ceph-cluster/1.12.7/defaults/cm.yaml
1663 https://artifacthub.io/packages/helm/bitnami/thanos/13.2.1
1664 https://artifacthub.io/packages/helm/bitnami/thanos/13.2.1?modal=values
1665 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/thanos/13.2.1/defaults/cm.yaml

Common Application Name	APP ID	Version	Component Versions	Helm Values	DKP Values
Traefik	traefik	26.0.0	• chart: 26.0.0[1666] • traefik: 2.10.6	Link[1667]	Link[1668]
Traefik ForwardAuth	traefik-forward-auth	0.3.11	• chart: 0.3.10[1669] • traefik-forward-auth: 3.1.0	Link[1670]	Link[1671]
Velero	velero	5.2.2	• chart: 5.2.2[1672] • velero: 1.12.3	Link[1673]	Link[1674]

13.2.7 DKP 2.8.1 Known Issues and Limitations

The following items are known issues with this release.

13.2.7.1 AWS Custom AMI Required for Kubernetes Version

Previous versions of DKP would default to using upstream AMIs published by the CAPA (Cluster API AWS) project when building AWS clusters if you did not specify your own AMI. However, those images are not currently available for the Kubernetes version used in the 2.8.1 release.

As a result, starting with this release of DKP, the behavior of the DKP `create cluster aws` command has been changed. It no longer defaults to using the upstream AMIs and instead requires that you specify an AMI built using Konvoy Image Builder (KIB) (see page 1626), or by explicitly requesting that it use the upstream images.

For more information on using a custom AMI in cluster creation or during the upgrade process, refer to these topics:

- AWS: Create the Management Cluster (see page 300)
- Enterprise: Upgrade the Management Cluster Kubernetes Version (see page 1735)

1666 https://artifacthub.io/packages/helm/traefik/traefik/26.0.0
1667 https://artifacthub.io/packages/helm/traefik/traefik/26.0.0?modal=values
1668 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/traefik/26.0.0/defaults/cm.yaml
1669 https://artifacthub.io/packages/helm/mesosphere/traefik-forward-auth/0.3.10
1670 https://artifacthub.io/packages/helm/mesosphere/traefik-forward-auth/0.3.10?modal=values
1671 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/traefik-forward-auth/0.3.11/defaults/cm.yaml
1672 https://artifacthub.io/packages/helm/vmware-tanzu/velero/5.2.2
1673 https://artifacthub.io/packages/helm/vmware-tanzu/velero/5.2.2?modal=values
1674 https://raw.githubusercontent.com/mesosphere/kommander-applications/v2.8.1/services/velero/5.2.2/defaults/cm.yaml

- Essential Upgrade Kubernetes Version (see page 1753)

13.2.7.2 vSphere and Ubuntu using Konvoy Image Builder Issue

Building a Konvoy Image Builder (KIB) image on vSphere using Ubuntu 20.04 with `cloud-init 23.3.3-0ubuntu0~20.04.1` will fail in the 2.8.1 version of DKP. The issue will be resolved in an upcoming KIB patch release.

13.2.7.3 Known CVE's

Since the release of DKP 2.6, DKP has been scanning Catalog applications for CVEs. Beginning with DKP 2.7, only the latest version of each Catalog application will be scanned (and have its critical CVEs mitigated). ` For more information about the known CVE's for compatible catalog apps, refer to D2iQ Security Updates (see page 1992).

Custom Banner on Cluster will not Save on Upgrade from DKP 2.7.0 to 2.8.1, if you have a custom banner it will no longer be available in the UI. You can then create a new banner.

13.2.8 DKP 2.8.1 Customer Incidents and Resolved Issues

The following resolved incidents are fixed or corrected in this release.

13.2.8.1 Dex DA Certificate Expiration Causes the UI to be Unreachable

A misconfiguration of the Dex certificate authority (CA) issuer and the `dex-client-tls` certificate resulted in the `dex-client-tls` certificate being invalid after approximately 90 days.
When the certificate becomes invalid, the DKP UI is no longer accessible and displays an "Internal Server Error - Failed to Retrieve Connector List" message.

The problem has been corrected.

D2IQ-100649

13.2.8.1.1 CAPA AMIs are Not Available for the Kubernetes Version used in DKP 2.8.1

AWS AMIs for some Kubernetes versions may not be available. We recommend using Konvoy Image Builder[1675] (KIB) to create a custom AMI.

- Do not use the upstream values as defaults for the `--ami-*` flags.
- If you specify any of these flags for AMI ID lookup, specify all of them: `--ami-base-os`, `--ami-format`, and `--ami-owner`.

See Also: DKP 2.8.1 Known Issues and Limitations (see page 1973)

D2IQ-100175

1675 https://d2iq.atlassian.net/wiki/spaces/DENT/pages/369393668

13.2.9 DKP 2.8.1 Deprecations

13.2.9.1 OS Deprecations

In the next release after 2.8.1, DKP will drop support for the following Operating Systems (OS):

- RHEL 7.9
- RHEL 8.4
- CentOS 7.9

For RHEL users, the recommendation is to upgrade to a supported version of RHEL 8.6 or RHEL 8.8.

For CentOS users, you can choose to migrate to any supported operating system in our support matrix found on the Supported Infrastructure Operating Systems (see page 67) page. We support Rocky Linux, RHEL and Ubuntu.

14 AI Navigator

First released in DKP 2.6.0, D2iQ engineered the AI Navigator with the power of AI. D2iQ then trained the chatbot on both our product documentation and Support knowledge base. This AI chatbot ensures real-time, interactive communication to answer a wide range of user queries, spanning basic instructions to complex functionalities.

Use of AI Navigator requires that you are logged in to a licensed copy of DKP version 2.6.0 or later, and that you accept the User Agreement (see page 1976). The AI chatbot is offered for Internet-connected environments regardless of DKP license type.

To help support heightened security in public sector, defense, and network-restricted customer environments, AI Navigator is **not** offered for air-gapped implementations.

14.1 AI Navigator User Agreement and Guidelines

Introducing the AI Navigator - engineered with the power of AI and models that include our product documentation and Support knowledge base, this chatbot ensures real-time, interactive communication to answer a wide range of user queries. The chatbot is offered for Internet-connected environments regardless of DKP license type.

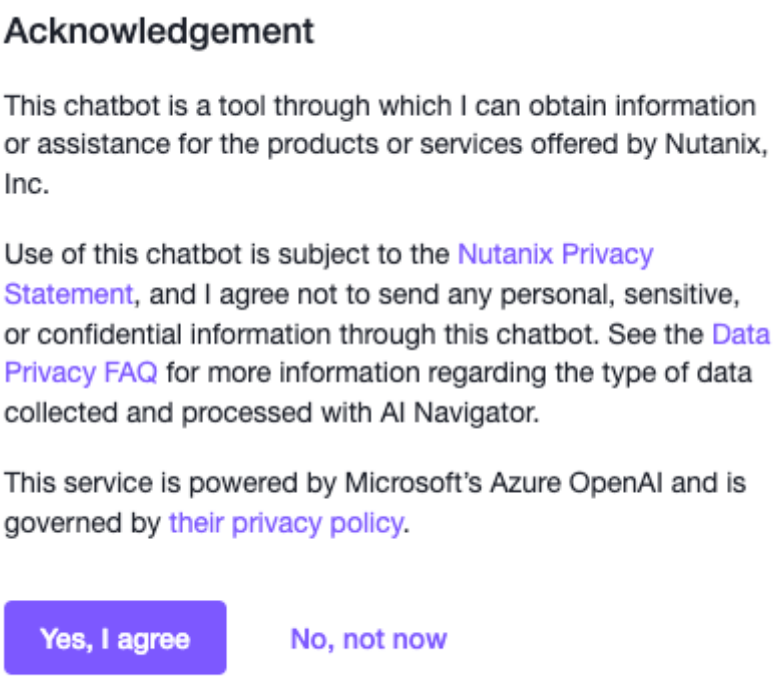
Use of the AI Navigator requires that you are logged in to a licensed copy of DKP 2.6.0 or later.

If you want to the AI Navigator to include cluster live data, additionally enable the DKP AI Navigator Cluster Info Agent: Obtain Live Cluster Information (see page 1986).

You can access the AI Navigator (see page 1979) application from the icon in the lower right corner of the DKP user interface.

14.1.1 Accept the User Agreement

When you first access the DKP AI Navigator, it displays the user agreement for acceptance:



Any use of the chatbot is pursuant to your acceptance of this agreement. If you select `No, not now` to decline, you cannot use AI Navigator.

14.1.2 Guidelines for Using AI Navigator

As with all AI engines, you should **NOT** enter sensitive information, including:

- Full names, addresses, or other personally identifying information (PII)
- Technical secrets, such as:
  - Actual server names
  - Real IP addresses
  - Private keys
  - Access tokens
  - …any other technical details that might be useful to malevolent actors
- Government ID numbers
- Corporate, financial, or investment information
- Company Confidential information
- Company credentials, including actual usernames or passwords
- Secured or classified information

…and so on.

> ⚠ Your company or organization may have specific rules about the kinds of prompts you should enter. Refer to your company or organization's specific guidelines for additional information.

You can read more here (see page 1981) on creating good prompts for use with AI Navigator.

14.1.2.1 Next Topic

AI Navigator in Installations and Upgrades (see page 1978)

14.2 AI Navigator in Installations and Upgrades

Use of AI Navigator requires that you are logged in to a licensed copy of DKP. The AI chatbot is offered for Internet-connected environments regardless of DKP license type.

> ⚠ To help support heightened security in public sector, defense, and network-restricted customer environments, AI Navigator is **not** offered for air-gapped implementations.

14.2.1 Install DKP with AI Navigator

When installing this version of DKP for the first time, you have a choice as to whether you install AI Navigator.

To **install** the chatbot application, perform the installation according to the procedures for your chosen infrastructure provider, and DKP installs AI Navigator by default.

To **disable** AI Navigator, modify the Kommander configuration file after generating it (see page 1538), to disable the `ai-navigator-app` entry in the `apps:` section of the file.

You can find more information about the Kommander configuration file in the topic, DKP Kommander Configuration Reference (see page 1560).

If you want to the AI Navigator to include cluster live data, enable the [Cluster Info Agent]: Obtain Live Cluster Information (see page 1986).

14.2.2 Upgrades to 2.7.0 and Later

By default, DKP installs the AI Navigator application as part of installing the Kommander component. DKP 2.6.0 installed AI Navigator by default as an experimental technical preview. However, beginning in DKP 2.6.1 and later, you have a choice to deactivate it. To **disable** AI Navigator, modify the Kommander configuration file after generating it (see page 1538), to disable the `ai-navigator-app` entry in the `apps:` section of the file.

14.2.2.1 For Upgrades in DKP Enterprise Environments

Under the DKP Enterprise upgrade (see page 0) section, expand the section for “For non-air-gapped environments.” Follow the instructions to add a CLI flag to the listed `dkp upgrade kommander` command to disable the chatbot.

14.2.2.2 For Upgrades in DKP Essential Environments

Under the DKP Essential upgrade (see page 0) section, expand the section for “For non-air-gapped environments.” Follow the instructions to add a CLI flag to the listed `dkp upgrade kommander` command to disable the chatbot.

14.2.2.3 Next Topic

Access the AI Navigator (see page 1979)

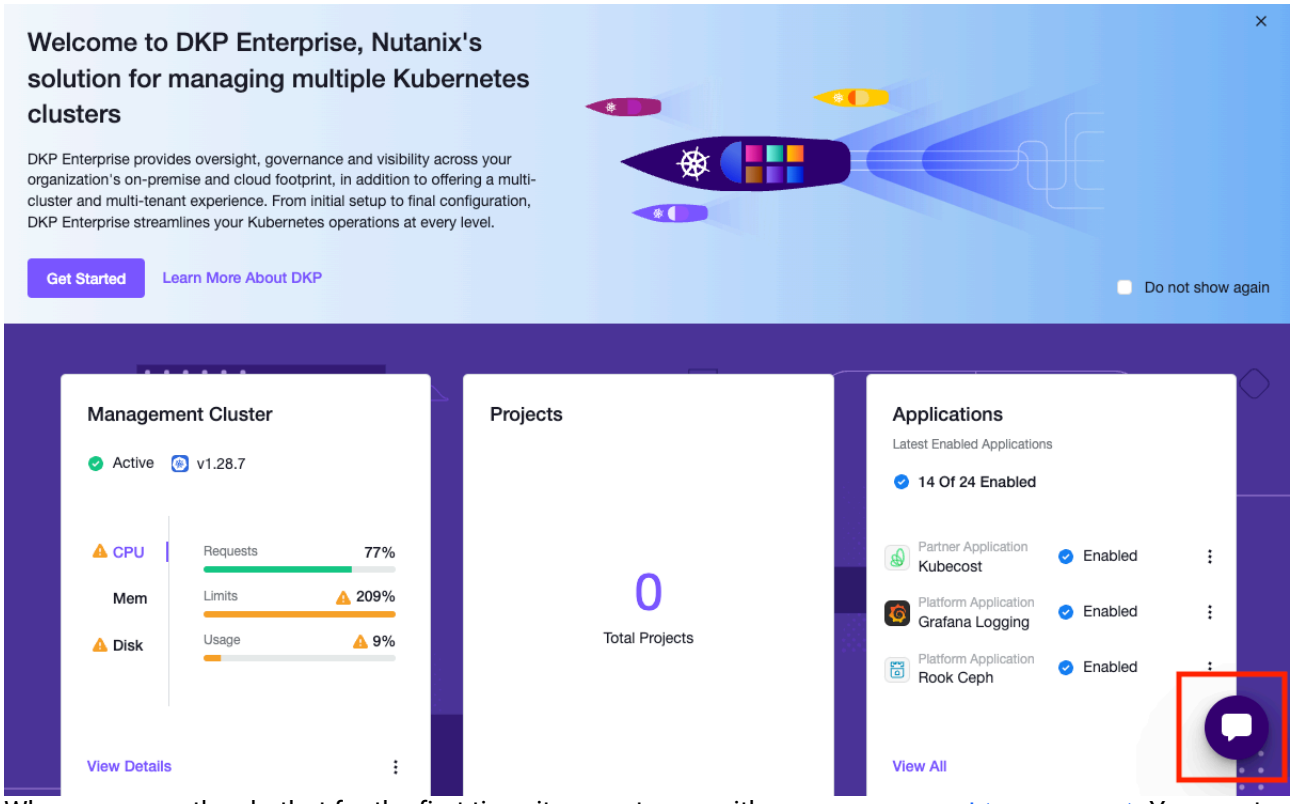
14.3 Access the AI Navigator

The chatbot is installed by default in Internet-connected environments, but you can choose not to install it.

> DKP does not offer the AI Navigator in air-gapped environments.

14.3.1 Open the AI Navigator

You can access the AI Navigator application from the icon in the lower right corner of the DKP user interface.

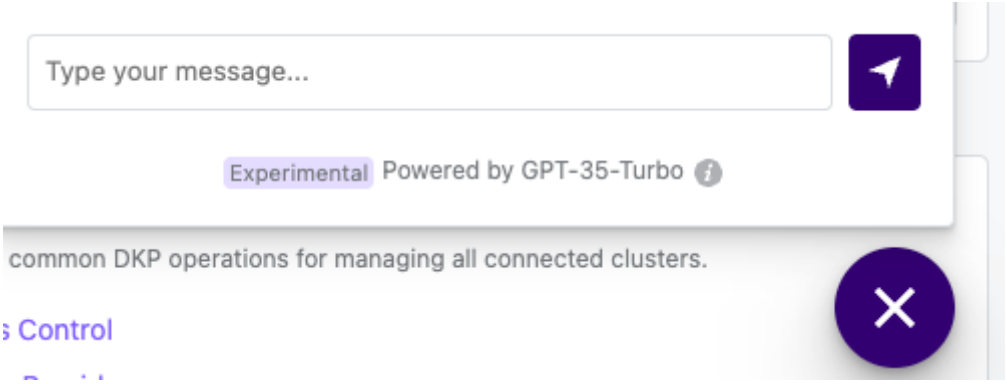


When you open the chatbot for the first time, it prompts you with a user agreement (see page 1976). You must agree to the terms to use AI Navigator. If you decline, the chatbot window closes and you cannot use it.

AI Navigator maintains your query history for the duration of your browser session, whether or not you close the AI Navigator application.

14.3.2 Close the AI Navigator

To close the AI Navigator, select the **Close** button just below the prompt entry field:



14.3.2.1 Next Topic

How to Create Good Queries (see page 1981)

14.4 How to Create Good Queries

Prompt engineering is the process of creating a chatbot that returns the data you really want. For AI chatbots trained on very large language models, there are some tips you can use to get better information with fewer prompt attempts. Some of those techniques are useful with the AI Navigator as well.

When you create a prompt, an AI chatbot breaks down your entry into discrete parts to help it search its model. The more precise you can make your prompt, the better the chatbot is at returning the information you wanted.

Another technique for getting to the desired information is called fine-tuning. This involves adjusting the parameters and database model that the chatbot has to search. DKP fine-tunes the AI Navigator model by adding both the DKP and Insights documentation, and the DKP Support Knowledge Base. This helps to keep answers focused and fast.

14.4.1 What Goes in a Prompt

You can include straight text, inline code snippets such as kubectl commands, and code block snippets, within a limit of 4096 characters.

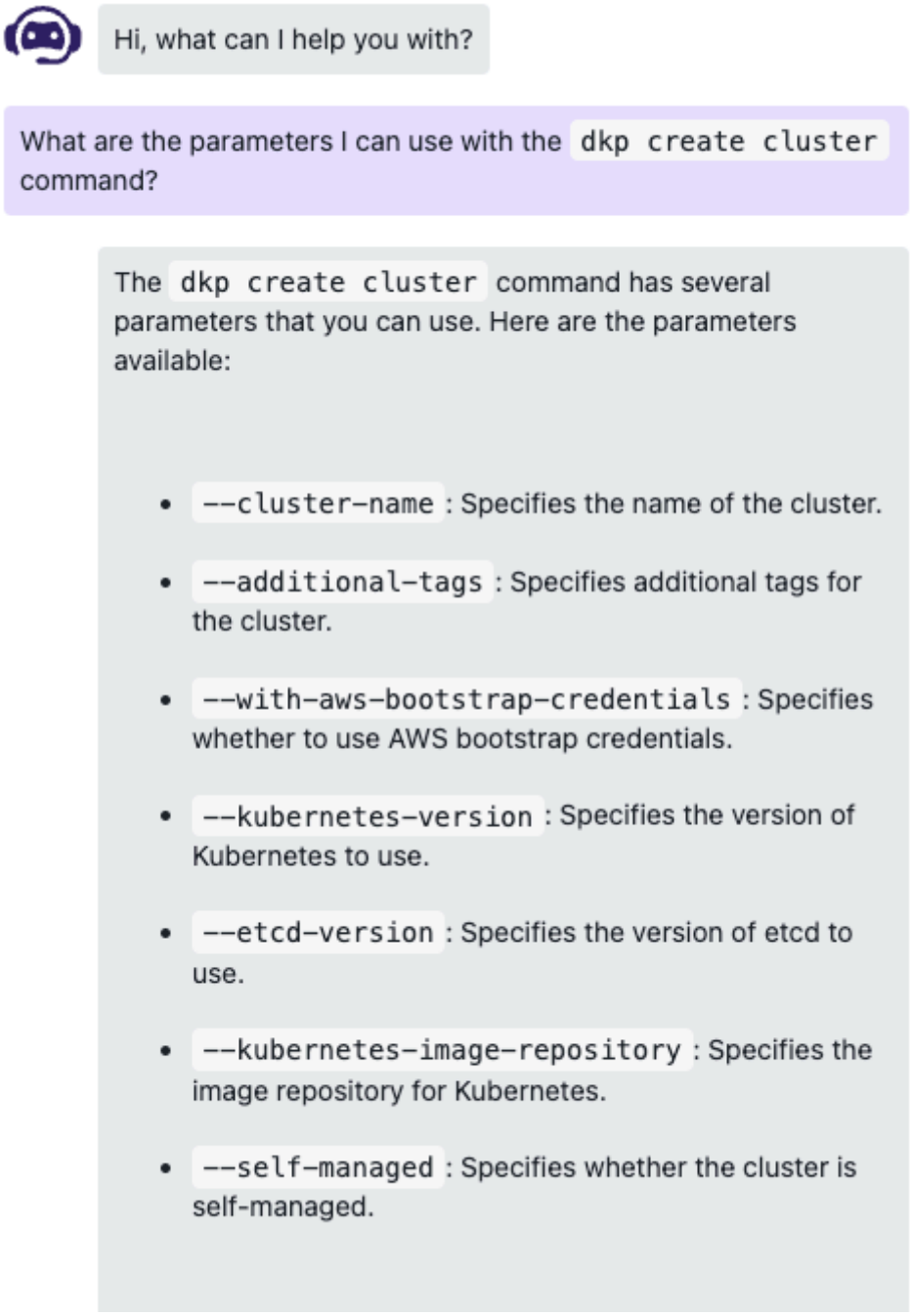
AI Navigator maintains your query history for the duration of your browser session, whether or not you close the AI Navigator application.

14.4.1.1 Inline Commands or Code Snippets

Use single back-tick marks to enclose a word, phrase, or command, for example:

```
What are the parameters I can use with the `dkp create cluster` command?
```

…creates this output in the chatbot window:



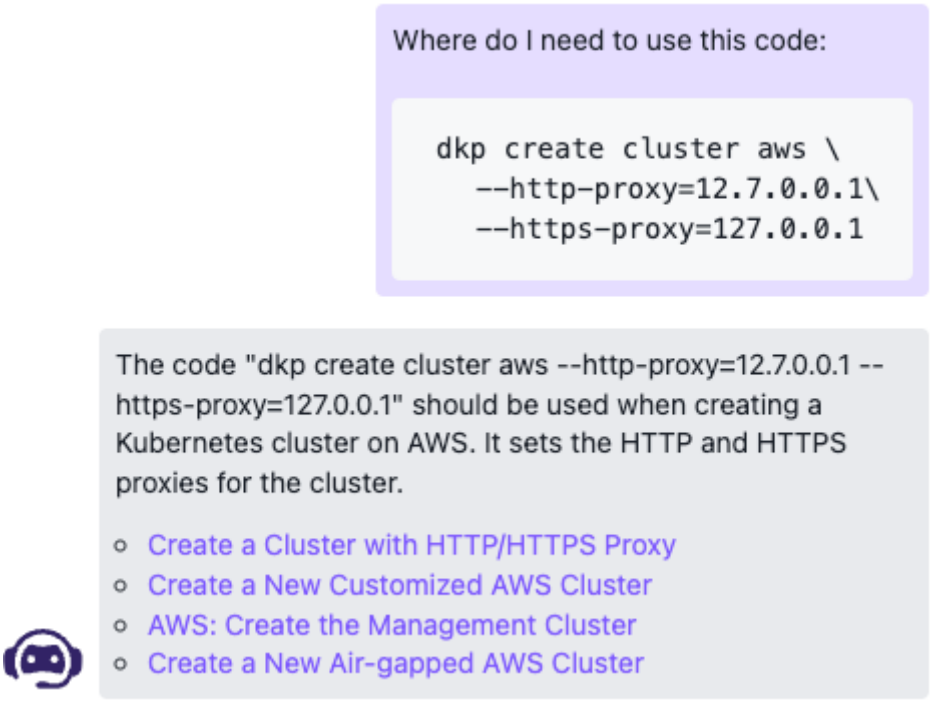
14.4.1.2 Code Blocks

Use a triple back-tick to open a code block for entry, in a manner similar to many popular chat and collaboration applications, like this:

```
Where do I need to use this code:
```

```
dkp create cluster aws \
    --http-proxy=127.0.0.1
    --https-proxy=127.0.0.2
```

...creates this entry and answer pair:



14.4.2 Selected Prompt Examples

You can use the AI Navigator to help you with a variety of tasks related to creating and managing clusters, and prompts can be fairly complex, and not necessarily limited to DKP. Consider these examples:

14.4.2.1 Evaluate a YAML Manifest

For this query:

```
Explain this yaml:

apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: nvidia-critical-app
spec:
  maxUnavailable: 1
  selector:
    matchLabels:
```

```
      app: nvidia-critical-app
```

...the AI Navigator returns output like this:

```
This YAML file is defining a Pod Disruption Budget (PDB) for an application named
"nvidia-critical-app".

A Pod Disruption Budget is a Kubernetes resource that specifies the minimum
number of pods that must be available during a disruption, such as a node
upgrade or maintenance. It helps ensure high availability of critical
applications by preventing too many pods from being unavailable at the same time.

In this specific YAML, the PDB is configured with the following properties:


maxUnavailable: 1: This specifies that only one pod of the "nvidia-critical-app"
can be unavailable at a time during a disruption. This ensures that at least
one pod is always available to handle requests.



selector: This specifies the labels used to select the pods that are part of
the "nvidia-critical-app". In this case, it selects the pods with the label
app: nvidia-critical-app.



Overall, this YAML is creating a Pod Disruption Budget that allows only one
pod of the "nvidia-critical-app" to be unavailable at a time during disruptions.

Links
-------
[Update Cluster Nodepools](https://beta-docs.d2iq.com/dkp/2.6/update-nvidia-gpu-
cluster-nodepools)
[Setting Priority Classes in DKP Applications](https://beta-docs.d2iq.com/dkp/2.6/
setting-priority-classes-in-dkp-applications)
[Project Quotas & Limit Ranges](https://beta-docs.d2iq.com/dkp/2.6/project-quotas-
limit-ranges)
[Configure the Control Plane](https://beta-docs.d2iq.com/dkp/2.6/configuring-the-
control-plane)
```

14.4.2.2 Updating a TLS Thumbprint

For this query:

```
How do I update a TLS Thumbprint?
```

…the AI Navigator returns output like this:

```
To update a TLS thumbprint in DKP, you need to update the following objects:

1. Update the vspherecluster object:

   - Use the command: kubectl patch vspherecluster <CLUSTER_NAME> --type=merge -p
'{"spec": {"thumbprint": "<TLS_THUMBPRINT>"}}'

2. Update the vsphere-config-secret secret in the vmware-system-csi namespace:

   - Use the command: kubectl patch secret vsphere-config-secret -n vmware-system-csi
--type='json'-p='[{"op" : "replace" ,"path" : "/data/csi-vsphere.conf" ,"value" :
"<BASE64 Encoded>"}]'

   - Make sure to encode the value of csi-vsphere.conf in base64.

3. Update the vsphere-cloud-config configmap in the kube-system namespace:

   - Use the command: kubectl --kubeconfig <CLUSTER_NAME>-workload.conf patch cm
vsphere-cloud-config -n kube-system --type=merge -p'{"data": {"vsphere.conf":
"global:\n  secretName: cloud-provider-vsphere-credentials\n  secretNamespace: kube-
system\n  thumbprint: <TLS_THUMBPRINT>\nvcenter:\n  <vCenter_Address>:\n
datacenters:\n    - 'dc1'\n    secretName: cloud-provider-vsphere-credentials\n
secretNamespace: kube-system\n    server: '<vCenter_Address>'\n    thumbprint:
<TLS_THUMBPRINT>\n" }}'


Replace <CLUSTER_NAME>, <TLS_THUMBPRINT>, and <BASE64 Encoded> with the actual values
specific to your environment.

Links
------
Updating vCenter Server TLS Certificate Thumbprint in DKP - D2iQ
Updating vCenter Server TLS Certificate Thumbprint in DKP - D2iQ
```

14.4.2.3 Describe Clock Skew in Ceph Cluster Status

For this query:

```
What is clock skew in Ceph cluster status?
```

…the AI Navigator returns output like this:

```
Clock skew refers to the difference in time between the clocks of different nodes in
a Ceph cluster. In the context of a Ceph cluster, clock skew can affect the monitors,
```

```
which are responsible for maintaining the cluster state. The monitors rely on a tight
time alignment to coordinate with other Ceph daemons. If there is a significant clock
skew between the monitor nodes, it can lead to issues in the cluster, such as
inconsistent cluster maps and coordination problems. It is important to keep the time
in-sync across nodes to ensure the proper functioning of the Ceph cluster.

Links
-------
How to detect if a clock skew is affecting Rook-Ceph monitors in DKP? - D2iQ
Create new cluster fails to complete due to time drift - D2iQ
```

14.5 DKP AI Navigator Cluster Info Agent: Obtain Live Cluster Information

Do you want to know which **etcd version** and **image** your cluster is using? Inquire about the frequency of a specific **CronJob**? Or just know which version of Insights[1676] you are using?

You no longer need to rely on your DKP and `kubectl` knowledge to obtain answers. By enabling the Cluster Info Agent, the AI Navigator can get the answers for you.

When enabled, the **Cluster Info Agent** can fetch live cluster data from your Essential or Management Cluster (see page 79), and help you find out more about your cluster and its deployments. Just start a new query with the AI Navigator. The answers will then include data related to your cluster's current status, allowing for enhanced specificity and context, and leading to quicker problem resolution and enhanced productivity.

Like the AI Navigator chatbot, the Cluster Info Agent is offered for Internet-connected environments regardless of DKP license type.

See Data Privacy FAQs [WIP] (see page 1988) for more information around data collection and processing.

To help support heightened security in public sector, defense, and network-restricted customer environments, AI Navigator is **not** offered for air-gapped implementations.

14.5.1 Cluster Info Agent Infrastructure

For the AI Navigator to be able to obtain live cluster information, it requires two components:

- `ai-navigator-info-api`: This is the collector of the application's API service, which performs all data abstraction data structuring services. This component is enabled by default and included in the AI Navigator.
- `ai-navigator-info-agent`: After manually enabling this platform application (see page 1987), the agent starts collecting Essential or Management cluster (see page 79) data and injecting it into the Cluster Info Agent database.

1676 https://d2iq.atlassian.net/wiki/spaces/DINS

14.5.1.1 Next Topic:

Enable, Customize, or Disable the DKP AI Navigator Cluster Info Agent (see page 1987)

14.5.2 Enable, Customize, or Disable the DKP AI Navigator Cluster Info Agent

14.5.2.1 Enable the DKP AI Navigator Cluster Info Agent

14.5.2.1.1 Prerequisites

- AI Navigator (see page 1978) is enabled
- Assign the `Workspace AI Navigator View Role` (`workspace-ai-navigator-cluster-info-view`) role to the user or user group that should have access to cluster-specific information when interacting with the AI Navigator. Refer to Configure Workspace Role Bindings (see page 711) for more information.

14.5.2.1.2 Enable the DKP AI Navigator Cluster Info Agent

Enable the Cluster Info Agent application on your Essential or Management cluster (see page 79) as explained in Enable an Application using the UI (see page 657).

- Customers with an Enterprise license must ensure they have selected the **Management Cluster Workspace** before deploying the Agent.
- The Cluster Info Agent requires a few hours to index the complete cluster dataset. Until then, you can still use the DKP AI Navigator, but it will not include the entire cluster-specific dataset when providing answers.

14.5.2.2 Customize the DKP AI Navigator Cluster Info Agent

The Cluster Info Agent automatically observes the `kommander` workspace namespace (that corresponds to the Essential or Management cluster (see page 79)). However, you can configure the agent to scan a broader context, for example, other namespaces, as long as they are in the same workspace.

To customize a AI Navigator Cluster Info Agent installation:

1. Log in to your DKP UI (see page 1556).
2. **Enterprise only:** Select the Management Cluster Workspace from the top navigation bar.

3. Select **Applications** from the sidebar and search for **AI Navigator Cluster Info Agent**.
4. Select the three-dot menu in the application card, and **Edit** > **Configuration**.
5. Add other namespaces for scanning, following this example:

```
watchNamespaces: namespace1,namespace2,namespace3
```

14.5.2.2.1 Uninstall the Cluster Info Agent

Uninstall the Cluster Info Agent application like any other application as explained in Disable an Application using the UI (see page 660).

14.5.2.2.2 Next Topic:

Data Privacy FAQs (see page 1988)

14.5.3 Data Privacy FAQs

What happens to my data during an AI Navigator query if the Cluster Info Agent is enabled?

D2iQ does not store any type of data related to your queries, nor Essential/Management cluster (see page 79) data.

Query-related data collected by the AI Navigator (with Cluster Info Agent enabled) is stored directly in your environment. Here is an overview of what happens during a query:

1. You make a query requesting information related to your live cluster.
2. AI Navigator leverages LangChain to retrieve query-relevant data from the Vector store.
3. AI Navigator sends relevant data from your cluster to our production service to be transmitted and processed by the Azure OpenAI Service.
4. The Azure OpenAI Service returns an answer to your query while including contextual cluster information.

What type of data is collected and processed?

- `nodes`
- `pods`
- `services`
- `events`
- `endpoints`
- `deployments`
- `statefulsets`
- `daemonsets`

- `replicasets`
- `ingresses`
- `jobs`
- `cronjobs`
- `helm.toolkit.fluxcd.io/helmreleases`

For which purpose is my cluster data being processed?

We prioritize your data's privacy and security. When you enable this feature, any potentially sensitive information from your Essential or Management cluster (see page 79) will be securely transmitted and processed by our trusted cloud service provider, Azure OpenAI. This data is used exclusively to add additional context to **your** queries. We adhere to strict data protection and privacy standards, ensuring responsible handling of your information.

Use of the AI Navigator chatbot with the Cluster Info Agent is governed by the terms of the D2iQ privacy policy[1677] and the Azure OpenAI privacy policy[1678].

Do I have control over where or how data is stored?

You cannot modify the way the chatbot processes or stores data. However, the AI Navigator Cluster Info Agent application is opt-in, which means that you can choose to enable or disable (see page 1987) it at your discretion.

Your informed consent is paramount, and we ensure clarity by offering a straightforward option to enable or disable this feature according to your preference.

1677 https://d2iq.com/privacy-policy
1678 https://learn.microsoft.com/en-us/legal/cognitive-services/openai/data-privacy

15 Access Documentation

The following sections describe how to access other versions or forms of our documentation.

15.1 Supported Documentation

You can access all n-2 supported documentation at D2iQ Help Center[1679].

15.2 Archived Documentation

In accordance with our version support policy[1680], we regularly archive older, unsupported versions of our documentation. At this time, this includes documentation for:

- DKP and DKP Insights 2.3 documentation at D2iQ Documentation Archive[1681].
- Konvoy, Kommander, Kaptain, DC/OS and Dispatch at our Public GitHub Repository[1682].

15.3 Download Documentation PDF

In an air-gapped environment, it is helpful to have a local PDF of the documentation. To download it, refer to the Download Documentation PDF (see page 1990) page.

15.4 Download Documentation PDF

This PDF contains the entire DKP documentation space. The file name contains the version and date created.

1679 http://docs.d2iq.com/
1680 https://docs.d2iq.com/dkp/latest/version-support-policy
1681 https://archive-docs.d2iq.com/
1682 https://github.com/mesosphere/dcos-docs-site/tags


Introducing the D2iQ Kubernetes Platform (DKP)

16 D2iQ Security Updates

View CVE security scans and update information for D2iQ products. The following information describes mitigated or fixed CVEs found in DKP. The "Vulnerability IDs" column contains the list of known mitigated CVE ids.

Product	Version	Vulnerability Reports	Vulnerability IDs
DKP	v2.8.0	Download CSV[1683]	Download File[1684]
DKP	v2.7.1	Download CSV[1685]	Download File[1686]
DKP	v2.7.0	Download CSV[1687]	Download File[1688]
DKP Catalog Applications	v2.7.0	Download CSV[1689]	Download File[1690]
DKP	v2.6.0	Download CSV[1691]	Download File[1692]
DKP Catalog Applications	v2.6.0	Download CSV[1693]	Download File[1694]
DKP	v2.5.0	Download CSV[1695]	Download File[1696]
DKP	v2.4.0	Download CSV[1697]	Download File[1698]

1683 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_185007_DKP_v2.8.0.csv
1684 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_185007_DKP_v2.8.0_ids.txt
1685 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_185005_DKP_v2.7.1.csv
1686 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_185006_DKP_v2.7.1_ids.txt
1687 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_185004_DKP_v2.7.0.csv
1688 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_185005_DKP_v2.7.0_ids.txt
1689 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_185005_DKP_Catalog_Applications_v2.7.0.csv
1690 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_185005_DKP_Catalog_Applications_v2.7.0_ids.txt
1691 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_185000_DKP_v2.6.0.csv
1692 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_185002_DKP_v2.6.0_ids.txt
1693 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_185003_DKP_Catalog_Applications_v2.6.0.csv
1694 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_185004_DKP_Catalog_Applications_v2.6.0_ids.txt
1695 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_184958_DKP_v2.5.0.csv
1696 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_185000_DKP_v2.5.0_ids.txt
1697 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_184953_DKP_v2.4.0.csv
1698 https://downloads.d2iq.com/dkp/cves/vulnerability_report_20240606_184957_DKP_v2.4.0_ids.txt

This report is generated using the scanner Trivy v0.44.0 and the CVE database is updated to 06 Jun 24 12:21 UTC.

16.1 CVE Policy

At D2iQ, our commitment to providing secure software solutions is paramount. We understand the critical importance of promptly addressing and mitigating security vulnerabilities to ensure the safety and trust of our customers and partners. This document outlines our policies and procedures regarding CVEs (Common Vulnerabilities and Exposures) to demonstrate our dedication to delivering software that is as secure as possible.

16.1.1 CVE Management:

Our procedure for managing CVE's is explained in the sections below.

16.1.1.1 Scanning Policy:

- Our primary objective is to deliver software that is free from critical security vulnerabilities (CVEs).
- To achieve this, we conduct regular scans of our software components, including:
  - Kubernetes
  - D2iQ Platform applications (Traefik, Istio, …)
  - D2iQ Catalog applications (*only* versions that are compatible with the default Kubernetes version supported with that DKP release, shown in our docs Workspace DKP Catalog Applications (see page 691))
  - DKP Insights Add-on
- Scans are performed every 24 hours using the latest CVE database to identify and address potential vulnerabilities promptly. The scanner version and the CVE database version are published among the security scan results.

16.1.1.2 Shipping Policy:

- Our goal is to ship software releases with no unmitigated Critical CVEs.
- For major and minor releases, we aim to ensure that there are no known, unmitigated critical CVEs.
- In the case of patch releases, if a critical CVE exists, it may be unmitigated if the only resolution involves upgrading a component to a new minor version.
- Critical CVEs in patch releases will be listed in the release notes' "Known Issues" section.
- We prioritize resolving these issues in the next minor release to maintain our commitment to security.
- In the event that we discover a critical CVE for a release that is already Generally Available (GA), and if the mitigation requires a patch release, we commit to releasing a patch with the CVE mitigation within a maximum of 60 days from the date of discovery. This timeframe ensures that our customers receive prompt attention and protection against critical vulnerabilities, even for GA releases.

16.1.1.3 **Known Issues:**

- If we become aware of a Critical CVE in any of our shipped applications, even if it is not (yet) mitigated, we will promptly add it to the D2iQ Security Updates page.
- For each CVE, we will provide mitigation steps that customers should take to protect their systems.
- We will also outline our plan for resolution in the "Mitigation" column of the report.

17 Version Support Policy

D2iQ® supports N-2 of the latest `MAJOR.MINOR` version of DKP. For example, if the current GA version of DKP® is 2.7.0, then D2iQ supports all patch versions of DKP 2.6.0, and 2.5.0.

When the next version releases, support continues for 2.7.x, and 2.6.x Support for DKP version 2.5.x expires. You should upgrade DKP with every new release to stay up-to-date with the latest features and bug fixes.

17.1 Supported Kubernetes Versions

Each DKP release supports a range of Kubernetes versions. Details for supported Kubernetes versions on DKP can be found in the Release Notes (see page 1946) on the Supported Kubernetes Version, as well as the Supported operating systems. Details for supported operating systems on DKP can be found in Supported Operating Systems (see page 67).

17.2 Features in Patches

Occasionally, to make new features available at a faster rate, D2iQ releases features as part of a patch release. If the Release Notes indicate a feature you need and do not yet have, consider upgrading to the latest version to take full advantage of new features and functions.

17.3 Experimental Status

“Experimental” means software, features, functionality, sample configurations, or other speculative content that is still under exploration, development, or testing by D2iQ. Experimental components carry no guarantee of eventual release as GA and therefore must not be used in Production Environments. Experimental components qualify for limited, Severity 4 support only and may be discontinued at any time, with or without notice.

Since Experimental components are not intended for Production Environment use, D2iQ cannot assume any responsibility for errors occurring during their use in Production. We can offer only these services in a commercially-reasonable manner, based on the availability of relevant subject-matter experts (SMEs):

- General operational guidance for the Experimental component.
- Identifying and diagnosing of errors in configuration or implementation, if possible.
- Advice on preventing and recovering from failures and troubleshooting, as available.

Support for Experimental components is provided on a Standard level, Severity 4 basis only.

This software is provided “as is” and without any express or implied warranties including, without limitation, the implied warranties of merchantability and fitness for a particular purpose.

17.4 Beta Feature

“Beta” feature means functionality that it is released to D2iQ’s customer base for testing and feedback before its official release(GA). Beta features are not considered fully complete or stable, and may contain known issues, gaps or limitations. The purpose of releasing a feature in Beta is to gather user feedback, identify bugs, gather insights on usability, and make necessary improvements before the feature is released General Availability (GA) as a stable and **production-ready** version. Beta features will receive a best effort support from D2iQ’s support & engineering teams with no SLA on response time. While it is our full intention to give a fantastic support experience, the early nature of the software may lead to longer support and fix times.

IMPORTANT: A Beta feature is:

- Not supported for use in Production Environments.
- Intended to eventually graduate to GA, though details may change.
- May not be supported for Upgrade to production GA.

17.5 End-of-Life (EoL)

In D2iQ, End-of-Life refers to versions of our software that are no longer fully supported. In relation to third party applications, EoL refers to applications that are not compatible with the latest version of Kubernetes that is supported in a release.

17.6 Enterprise OS - Red Hat Enterprise Linux (RHEL) Support Policy

As part of our commitment to delivering the best possible experience to our customers, D2iQ aligns its support for RHEL versions based on their Extended Update Support.[1699] We prioritize supporting RHEL minor versions with Extended Update Support, ensuring customers can benefit from long-lasting stability and security.

For each DKP version, we will ensure compatibility and support with the latest released RHEL 8 minor version that has Extended Update Support at the start of the DKP version's development phase. This approach allows us to maintain a seamless integration with the most secure and well-supported RHEL releases, enabling our customers to confidently operate their DKP environments with optimal reliability and peace of mind.

1699 https://access.redhat.com/articles/rhel-eus

17.7 Support Definitions

17.7.1 Secondary Support

The following section describes D2iQ's support for secondary applications, such as platform applications. All platform applications that D2iQ ships with DKP products are covered under secondary support:

Type	Scope Example	Support Offered
Configuration	• Guidance for base technology and DKP interoperability configuration questions and troubleshooting for different components of the DKP platform. • No support for base technology's configuration that is unrelated to its integration with DKP. • No support for performance issues with the base technology that is unrelated to its integration with DKP.	Supported with severity 3 & 4 support terms

Type	Scope Example	Support Offered
Failure Assistance	• Assistance with installation, and upgrade failures of the service as captured in the supported DKP product upgrade pathway. • Assistance with service failures due to platform issues. For example: DKP Enterprise or DKP Essential • Support is limited to troubleshooting for root cause up to DKP product limit. Root causes that are identified to be beyond this limit will need to be pursued by the company who creates the platform application base technology. Note, if the RCA for the failure is due to a non-standard configuration or non-DKP use of the platform application, we will be unable to provide assistance beyond basic identification. • No assistance for base technology's failures that is unrelated to its integration with DKP.	Supported with all severities
Bug Fixes	• Bug fixes for service integration with DKP. • Upstream bug fixes to identified issues in the base technology of the platform application on a best effort basis. • No guarantees that our changes to upstream will be accepted. • No commitment to maintaining forks upstream.	RCA supported with all severities, Fix supported with severity 3 & 4 support terms

Type	Scope Example	Support Offered
Documentation errors	• Documentation fixes for life cycle management of services and integration with DKP. • Upstream documentation fixes to reported and identified issues in base technology of the platform application via a best effort basis. • No guarantees that our changes will be accepted.	Supported with severity 4 support terms

17.8 Standard Level & Severity Definitions

To view our severity level and support terms, refer to D2iQ Support and Maintenance Terms[1700].

17.9 Operating System (OS) Version Support Policy

This page contains details about D2iQ's support policy in relation to different operating systems.

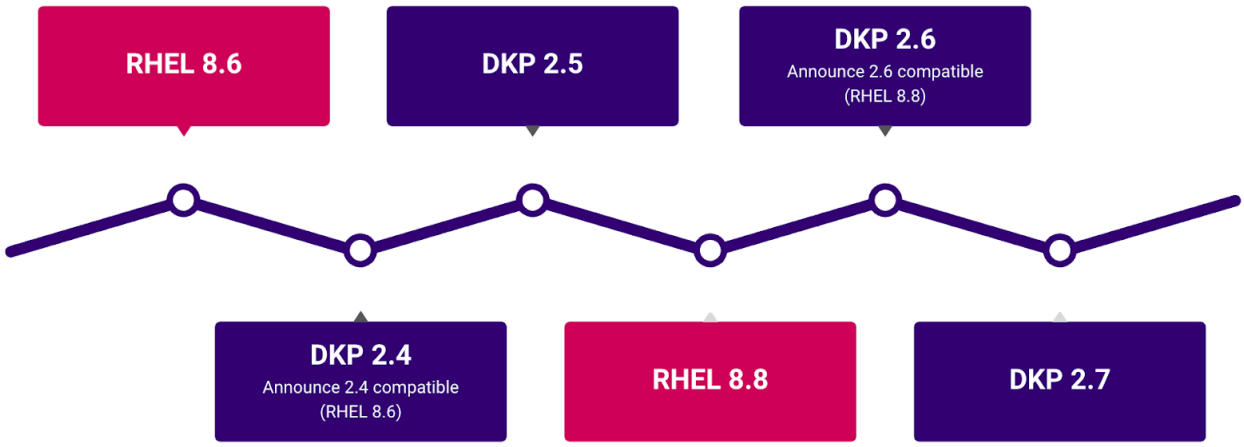
17.9.1 Red Hat Enterprise Linux (RHEL) Version Support Policy

As part of our commitment to delivering the best possible experience to our customers, D2iQ aligns its support for RHEL versions based on their Extended Update Support.[1701] We prioritize supporting RHEL minor versions with Extended Update Support, ensuring customers can benefit from long-lasting stability and security.

For each DKP version, we will ensure compatibility and support with the latest released RHEL 8 minor version that has Extended Update Support at the start of the DKP version's development phase. This approach allows us to maintain a seamless integration with the most secure and well-supported RHEL releases, enabling our customers to confidently operate their DKP environments with optimal reliability and peace of mind.

1700 https://d2iq.com/legal/support-terms
1701 https://access.redhat.com/articles/rhel-eus

17.9.1.1 Example:



18 Legal Notices

D2iQ® and its licensors are the owners of all right, title, and interest in and to D2iQ software products, the documentation, all updates, upgrades, and derivative works thereto, and all intellectual property rights therein. D2iQ software products may additionally include third-party open source software.

See the D2iQ Legal page[1702] for additional details.

1702 https://d2iq.com/legal/3rd