Oracle Cloud Native Environment

Upgrade to Release 2

F96199-04
August 2025

Oracle Cloud Native Environment Upgrade to Release 2,

F96199-04

Contents

Preface

This document contains information about upgrading Oracle Cloud Native Environment (Oracle CNE) Release 1 to Release 2.

1 Introduction

Introduces upgrading from Oracle Cloud Native Environment (Oracle CNE) Release 1 to Release 2.

> **Important**
>
> The upgrade from Oracle CNE Release 1 to Release 2 can be done with Oracle CNE Release 1.9. If you're using an earlier version of Oracle CNE, upgrade to Release 1.9 before you perform the upgrade to Release 2.
>
> You must be running the latest available errata release of Oracle CNE Release 1.9, and all installed modules must be updated, before you perform the upgrade to Release 2. For information on upgrading to the latest errata release, see Oracle Cloud Native Environment: Updates and Upgrades for Release 1.9.
>
> Upgrading requires a highly available Release 1 cluster. You can't upgrade single node clusters, or clusters with less than three control plane nodes.

Several upgrade paths are available. Some require access to the system's console to inject a Kickstart file at boot time. Other paths don't require this, and instead use the option of replacing the boot disk using the Oracle Linux Virtualization Manager or OCI User Interface. Use the following table to decide which upgrade path is most appropriate.

Table 1-1 Server Upgrade Options

Server Type	Replaces Boot Disk	Requires System Boot Console Access
Bare metal systems		Upgrade a Bring Your Own Cluster
Oracle Linux Kernel-based Virtual Machine (KVM) instances		Upgrade a Bring Your Own Cluster
Oracle Linux Virtualization Manager Virtual Machines (VMs)	Upgrade an Oracle Linux Virtualization Manager Cluster	Upgrade a Bring Your Own Cluster
OCI instances	Upgrade an OCI Cluster	
Oracle Private Cloud Appliance virtual instances		Upgrade a Bring Your Own Cluster
Oracle Private Cloud at Customer virtual instances		Upgrade a Bring Your Own Cluster

This is an in-place migration, where nodes are updated to use the Oracle CNE Release 2 OS. At the end of the migration:

- All cluster nodes have the Oracle CNE Release 2 OS installed.

- The Oracle CNE CLI can manage all nodes.
- The nodes are running the latest version of Kubernetes, as stated in Oracle Cloud Native Environment: Release Notes.
- All Release 1 modules are available as applications in the Oracle application catalog. All Release 1 Kubernetes applications are also available in the catalog.

The upgrade is performed using the Bring Your Own (BYO) provider. Before you begin, we recommend you read about the BYO provider in Oracle Cloud Native Environment: Concepts and Oracle Cloud Native Environment: Kubernetes Clusters to understand the architecture of a BYO cluster, and of Oracle CNE Release 2.

2 Prerequisites

Set up the required prerequisites to upgrade from Oracle CNE Release 1 to Release 2.

The upgrade is performed on an Oracle Linux host that's outside the Release 1 cluster. This host must have network access to the Kubernetes nodes in the Release 1 cluster.

> **Important**
>
> The upgrade must not be performed on any of the Release 1 Kubernetes nodes. The upgrade steps must be performed on a host outside the cluster.

Before you begin, identify any OS Customizations in the Release 1 cluster. If you have OS customizations that can't be set up using a configuration file, you might need to use Oracle Container Host for Kubernetes Image Builder to build a custom image to use during the upgrade.

To prepare for the upgrade, on an Oracle Linux host, set up:

- The Oracle CNE Command Line Interface (CLI), `ocne`.
- The Oracle CNE libvirt provider. This is used to create temporary Kubernetes clusters to perform some steps in the upgrade.
- The Kubernetes CLI, `kubectl`.
- A copy of the Release 1 Kubernetes configuration file, `kubeconfig`.

OS Customizations

The upgrade from Oracle CNE Release 1 to Release 2 uses the Bring Your Own (BYO) provider. The image used for the BYO provider is based on OSTree. The steps in this guide use two methods of upgrade, either installing the OS image on an system's existing disk (using the Anaconda and Kickstart installation options of Oracle Linux), or swapping a virtual disk with one that contains the OS image.

The Oracle CNE Release 2 OS image doesn't provide the option to install or configure the OS after it's deployed. The OS image is immutable and can't be changed after it's installed. Any customizations to the OS must be done in the image used for the installation of Release 2. Any updates to the OS must be done using a new OSTree based image, which is how OS updates are performed.

If you have any customized OS requirements on the systems in the Oracle CNE Release 1 cluster, you need to identify those before you begin the upgrade. For example, if you have any software packages and OS configuration that's in addition to the standard requirements in the Release 1 deployment.

If any customizations are required, you might have the option to set these in a Release 2 cluster configuration file. If configuration file options are available for the customization, ensure you add these to a cluster configuration file before you upgrade. For information on the

available cluster file configuration options, see Oracle Cloud Native Environment: Kubernetes Clusters.

If customization settings aren't available in a cluster configuration file option, you might need to use Oracle Container Host for Kubernetes Image Builder to build a custom image. For information on building a custom image, see Oracle Cloud Native Environment: Oracle Container Host for Kubernetes Image Builder.

Installing the CLI

Install the Oracle Cloud Native Environment (Oracle CNE) Command Line Interface (CLI) on an Oracle Linux host, using the Oracle Linux Yum Server, or the Unbreakable Linux Network (ULN).

The Oracle CNE CLI is the command line tool to create and manage Kubernetes clusters in Oracle CNE. The CLI (`ocne` command) includes a help system to show all command options, and a set of configuration files at various levels to configure the environment and clusters.

1. Set up the Oracle Linux Yum Server Repository.

   If the system uses the Oracle Linux Yum Server, set it up to install the CLI:

   Oracle Linux 9:

   ```
   sudo dnf install -y oracle-ocne-release-el9
   sudo dnf config-manager --enable ol9_ocne
   ```

   Oracle Linux 8:

   ```
   sudo dnf install -y oracle-ocne-release-el8
   sudo dnf config-manager --enable ol8_ocne
   ```

2. Set up ULN.

   If the system is registered to use ULN, use the ULN web interface to subscribe the system to the appropriate channel.

   For Oracle Linux 9, subscribe to `ol9_x86_64_ocne` or `ol9_aarch64_ocne`.

   For Oracle Linux 8, subscribe to `ol8_x86_64_ocne` or `ol8_aarch64_ocne`.

3. Install the CLI.

   ```
   sudo dnf install -y ocne
   ```

Setting Up the libvirt Provider

Set up an Oracle Linux host to create Kubernetes clusters using the `libvirt` provider.

Clusters can be created on the localhost, or on a remote system. Perform these steps on the system to be used to create the cluster, whether that's the localhost for local clusters, or on a remote host if you're creating clusters on a remote system.

By default, KVM is built into the Oracle Linux kernel. You can use the default KVM stack, which includes libvirt. We recommend you use the Oracle KVM stack which is available in Oracle Linux 8 or 9 with the Unbreakable Enterprise Kernel (UEK). For Oracle Linux 9, the latest UEK Release 7 (UEK R7) must be installed. For Oracle Linux 8, UEK R6 or UEK R7 must be installed.

> **Important**
>
> Existing Virtual Machines created with one KVM stack might not be compatible, and might not start, after switching to another KVM stack.

For more information on installing and configuring KVM, see the Oracle Linux 9: KVM User's Guide or the Oracle Linux 8: KVM User's Guide.

1. (Optional) Install the Oracle KVM stack.
   - Oracle Linux 9:

     If you have an existing installation of the default KVM stack, remove it:

     ```
     sudo dnf remove -y libvirt qemu-kvm edk2
     ```

     Install the Oracle KVM stack:

     ```
     sudo dnf config-manager --enable ol9_kvm_utils
     sudo dnf group install -y "Virtualization Host"
     sudo dnf install -y virt-install virt-viewer
     ```

     Start the virtualization daemons.

     ```
     for drv in qemu network nodedev nwfilter secret storage interface
     proxy
       do
        sudo systemctl enable virt${drv}d.service
        sudo systemctl enable virt${drv}d{,-ro,-admin}.socket
        sudo systemctl start virt${drv}d{,-ro,-admin}.socket
       done
     ```

   - Oracle Linux 8:

     If you have an existing installation of the default KVM stack, remove it:

     ```
     sudo dnf module remove -y virt --all
     sudo dnf module reset virt
     ```

     Install the Oracle KVM stack:

     ```
     sudo dnf config-manager --enable ol8_kvm_appstream
     sudo dnf module enable virt:kvm_utils3
     sudo dnf module install -y virt:kvm_utils3
     ```

     Enable and start the `libvirtd.service`:

     ```
     sudo systemctl enable --now libvirtd.service
     ```

2. Validate the host.

Validate the host is set up for hardware virtualization, and can be used as a KVM host:

```
virt-host-validate qemu
```

3. Configure the user.

   Configure the user to have privileged access to libvirt, add the user to the `libvirt` and `qemu` groups.

   ```
   sudo usermod -a -G libvirt,qemu $USER
   ```

   To enable the change to the user, log out, and log back into the host or terminal session.

4. (Optional) Open a range of ports in the firewall.

   If you're installing libvirt on a remote host, open a series of firewall ports so you can access nodes in the cluster from the localhost. You don't need to do this if you're installing libvirt on the localhost. Use the format:

   ```
   sudo firewall-cmd --add-port 6443-endrange/tcp
   sudo firewall-cmd --add-port 6443-endrange/tcp --permanent
   ```

   Replace *endrange* with the highest port number you want to open. For example, to open 20 ports, use:

   ```
   sudo firewall-cmd --add-port 6443-6463/tcp
   sudo firewall-cmd --add-port 6443-6463/tcp --permanent
   ```

   Restart `firewalld.service`

   ```
   sudo systemctl restart firewalld.service
   ```

Installing kubectl

Install the Kubernetes CLI, `kubectl`.

Install `kubectl` on the localhost (the host with `ocne` installed):

```
sudo dnf install kubectl
```

Adding the Release 1 Kubernetes Configuration File

Save the Kubernetes configuration file for the Release 1 cluster on the localhost.

The Kubernetes configuration file for the Oracle Cloud Native Environment Release 1 cluster must be available on the localhost (the host with `ocne` installed).

> **Important**
>
> The Kubernetes configuration file **must** be saved as the name of the Release 1 cluster (the Kubernetes Module name).

Copy the `kubeconfig` file for the Release 1 cluster to the localhost as:

`$HOME/.kube/kubeconfig.`*`cluster_name`*

Where *`cluster_name`* is the name of the Release 1 cluster. For example:

```
$HOME/.kube/kubeconfig.mycluster
```

3

Prepare the Release 1 Cluster

Prepare the Oracle CNE Release 1 cluster to perform an upgrade to Release 2.

Before you perform the upgrade:

- Back up the Release 1 cluster.
- Install the Release 2 application catalog and User Interface (UI) into the Release 1 cluster.
- If Calico is installed, update it.

Backing Up the Cluster

Back up the Oracle CNE Release 1 cluster.

In the Release 1 cluster, access the operator node and use the `olcnectl module backup` command to back up the Kubernetes Module in the environment. Use the syntax:

```
olcnectl module backup \
--environment-name environment_name \
--name cluster_name
```

For example:

```
olcnectl module backup \
--environment-name myenvironment \
--name mycluster
```

For more information on backing up a Release 1 cluster, see Oracle Cloud Native Environment: Kubernetes Module for Release 1.9.

Installing the Application Catalog and UI

Install the Oracle CNE application catalog and User Interface (UI) into the Release 1 cluster.

It might be helpful to show the applications in the Oracle CNE Release 1 cluster are added to the application catalog when you install it into a cluster. Any modules and applications that are running are added to the application catalog.

1. Install the UI and application catalog.

   Use the `ocne cluster start` with the `--provider` option set to `none` and specify the location of the Release 1 cluster's `kubeconfig` file. The syntax is:

   ```
   ocne cluster start
   [{-u|--auto-start-ui} {true|false}]
   [{-o|--boot-volume-container-image} URI]
   [{-C|--cluster-name} name]
   [{-c|--config} path]
   [{-n|--control-plane-nodes} integer]
   ```

```
[{-i|--key} path]
[--load-balancer address]
[{-P|--provider} provider]
[{-s|--session} URI]
[{-v|--version} version]
[--virtual-ip IP]
[{-w|--worker-nodes} integer]
```

For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

For example:

```
ocne cluster start --provider none --kubeconfig $HOME/.kube/
kubeconfig.mycluster
```

2. (Optional) Configure access to the UI.

   Follow the prompts in the output of the `ocne cluster start` command to set up an access token for the UI. For more information on setting up an access token and connecting to the UI, see Oracle Cloud Native Environment: Kubernetes Clusters.

3. Verify the Oracle catalog is installed.

   Use the `ocne catalog list` command to verify the Oracle application catalog is installed. The syntax is:

   ```
   ocne catalog {list|ls}
   ```

   For example:

   ```
   ocne catalog list
   ```

4. Verify the list of available applications.

   Use the `ocne catalog search` command to see the list of applications available in the Oracle catalog. The syntax is:

   ```
   ocne catalog search
   [{-N|--name} name]
   [{-p|--pattern} pattern]
   ```

   To list all the applications available in the Oracle catalog:

   ```
   ocne catalog search
   ```

Updating the Calico Module

If Calico is deployed in the Release 1 cluster as the Calico Module, it must be updated so it runs correctly in the Release 2 cluster. This task must be completed before you continue with the upgrade, or the Kubernetes cluster can't start up when being upgraded.

> **Note**
>
> Calico can be installed in Release 1 as either the Kubernetes Container Network Interface (CNI), or using the Calico Module. If Calico is deployed as the Kubernetes CNI, update Calico using the steps in Updating Calico as the Kubernetes CNI. For information on the Calico installation methods in Release 1.9, see Oracle Cloud Native Environment: Calico Module for Release 1.9.

This task removes the existing Calico Tigera operator, and installs it again from the Release 2 application catalog. We recommend you install the latest available version of the Calico Tigera operator listed in the application catalog, which might be a newer release than shown here.

1. Search the application catalog for Calico Tigera operator versions.

   Several Calico Tigera operator versions are available to install. We recommend you install the latest version. To see the list available versions, use:

   ```
   ocne catalog search --pattern tigera-operator
   ```

2. Get information about the Calico Tigera operator.

   Save the information about the Calico Tigera operator to a file. Use:

   ```
   ocne application show --namespace tigera-operator --release
   calico_module_name > filename.yaml
   ```

   Replace *calico_module_name* with the name of Calico module, and filename with the *filename* to use for the configuration file. For example:

   ```
   ocne application show --namespace tigera-operator --release mycalico >
   calico.yaml
   ```

   The output file contains information about the Calico Tigera operator. You use this information to create a Helm configuration file for installing the Calico Tigera operator from the application catalog.

3. Create a Helm configuration file.

   Edit the YAML file created in the previous step. Delete the first few lines that contain the `calico` section, for example:

   ```
   calico:
     releasename: mycalico
     version: 3.27.5
   ```

   Include a line for `flexVolumePath: /var/lib/kubelet/volumeplugins/` entry in the `installation` section:

   ```
   installation:
    flexVolumePath: /var/lib/kubelet/volumeplugins/
   ```

Include the `version` number in the `tigeraOperator` section. We recommend you use the latest version of the Tigera Calico operator that's available in the application catalog, for example:

```
tigeraOperator:
  version: v1.32.4
```

Remove the last part of the file after the dashes, for example:

```
---------------
NAME: mycalico
NAMESPACE: tigera-operator
CHART: calico
STATUS: deployed
REVISION: 1
APPVERSION: 3.27.5
```

The resulting file might look similar to:

```
calicoctl:
  registry: container-registry.oracle.com/olcne
installation:
  flexVolumePath: /var/lib/kubelet/volumeplugins/
  calicoNetwork:
    bgp: Disabled
    ipPools:
    - cidr: 10.244.0.0/16
      encapsulation: VXLAN
  cni:
    type: Calico
  imagePath: olcne
  registry: container-registry.oracle.com
tigeraOperator:
  version: v1.32.4
  namespace: tigera-operator
  registry: container-registry.oracle.com/olcne
```

4. Uninstall the Tigera Calico operator.

   ```
   ocne application uninstall --namespace tigera-operator --release
   calico_module_name
   ```

5. Install the Tigera Calico operator from the application catalog.

   ```
   ocne application install --namespace tigera-operator --release calico --
   values filename.yaml --name tigera-operator
   ```

   Where *filename* is the name of the Helm configuration file.

6. Validate the Tigera Calico operator installation.

   ```
   kubectl describe pod --namespace tigera-operator tigera-operator
   ```

Enter the **Tab** key to auto complete the name of the `tigera-operator` pod.

The output contains the `Image` option, which shows the Tigera Operator version, is now `1.32.4`. For example:

```
    Image:         container-registry.oracle.com/olcne/tigera-
operator:v1.32.4
```

Check the application configuration:

```
ocne application show --namespace tigera-operator --release calico
```

The output contains the information used in the configuration file to set up the application. For example:

```
calicoctl:
  registry: container-registry.oracle.com/olcne
installation:
  calicoNetwork:
    bgp: Disabled
    ipPools:
    - cidr: 10.244.0.0/16
      encapsulation: VXLAN
  cni:
    type: Calico
  flexVolumePath: /var/lib/kubelet/volumeplugins/
  imagePath: olcne
  registry: container-registry.oracle.com
tigeraOperator:
  namespace: tigera-operator
  registry: container-registry.oracle.com/olcne
  version: v1.32.4
----------------
NAME: calico
NAMESPACE: tigera-operator
CHART: tigera-operator
STATUS: deployed
REVISION: 1
APPVERSION: 1.32.4
```

Check the Tigera Calico operator status using:

```
kubectl get tigerastatus
```

The output shows the status of the Tigera Calico operator components.

Updating Calico as the Kubernetes CNI

If Calico is deployed in the Release 1 cluster as Kubernetes CNI, it must be updated so it runs correctly in the Release 2 cluster. This task must be completed before you continue with the upgrade, or the Kubernetes cluster can't start up when being upgraded.

This task updates the existing Calico Tigera operator, and installs it again from the Release 2 application catalog. We recommend you install the latest available version of the Calico Tigera operator listed in the application catalog, which might be a newer release than shown here.

1. Update the Calico Tigera operator.

   You need to set a release name for the Calico Tigera operator. This example uses `calico`. Change the `release-name=calico` entries in this set of commands to use a difference release name for the application. Run the following commands to update the existing operator:

   ```
   kubectl -n tigera-operator label ServiceAccount tigera-operator
   app.kubernetes.io/managed-by=Helm
   kubectl -n tigera-operator annotate ServiceAccount tigera-operator
   meta.helm.sh/release-name=calico
   kubectl -n tigera-operator annotate ServiceAccount tigera-operator
   meta.helm.sh/release-namespace=tigera-operator

   kubectl -n tigera-operator label ClusterRole tigera-operator
   app.kubernetes.io/managed-by=Helm
   kubectl -n tigera-operator annotate ClusterRole tigera-operator
   meta.helm.sh/release-name=calico
   kubectl -n tigera-operator annotate ClusterRole tigera-operator
   meta.helm.sh/release-namespace=tigera-operator

   kubectl -n tigera-operator label ClusterRoleBinding tigera-operator
   app.kubernetes.io/managed-by=Helm
   kubectl -n tigera-operator annotate ClusterRoleBinding tigera-operator
   meta.helm.sh/release-name=calico
   kubectl -n tigera-operator annotate ClusterRoleBinding tigera-operator
   meta.helm.sh/release-namespace=tigera-operator

   kubectl -n tigera-operator label Deployment tigera-operator
   app.kubernetes.io/managed-by=Helm
   kubectl -n tigera-operator annotate Deployment tigera-operator
   meta.helm.sh/release-name=calico
   kubectl -n tigera-operator annotate Deployment tigera-operator
   meta.helm.sh/release-namespace=tigera-operator

   kubectl label APIServer default app.kubernetes.io/managed-by=Helm
   kubectl annotate APIServer default meta.helm.sh/release-name=calico
   kubectl annotate APIServer default meta.helm.sh/release-namespace=tigera-
   operator

   kubectl label Installation default app.kubernetes.io/managed-by=Helm
   kubectl annotate Installation default meta.helm.sh/release-name=calico
   kubectl annotate Installation default meta.helm.sh/release-
   namespace=tigera-operator
   ```

2. Search the application catalog for Calico Tigera operator versions.

   Several Calico Tigera operator versions are available to install. We recommend you install the latest version. To see the list of available versions, use:

   ```
   ocne catalog search --pattern tigera-operator
   ```

3. Install the Tigera Calico operator from the application catalog.

For example:

```
ocne application install --namespace tigera-operator --release calico --
version 1.32.4 --name tigera-operator
```

The `--release` option must be set to the same name used for `release-name` in the first step. In this example, this is `calico`.

4. Validate the Tigera Calico operator installation.

```
kubectl describe pod --namespace tigera-operator tigera-operator
```

Enter the **Tab** key to auto complete the name of the `tigera-operator` pod.

The output contains the `Image` option, which shows the Tigera Operator version, is `1.32.4`. For example:

```
    Image:         container-registry.oracle.com/olcne/tigera-
operator:v1.32.4
```

Check the Tigera Calico operator status using:

```
kubectl get tigerastatus
```

The output shows the status of the Tigera Calico operator components.

4

Upgrade a Bring Your Own Cluster

Upgrade a Kubernetes cluster using the Bring Your Own (BYO) provider. Use this to upgrade a cluster from Oracle CNE Release 1 to Release 2.

The steps to upgrade a Release 1 cluster to Release 2 are:

1. Create a Release 2 cluster configuration file.
2. Create an OSTree image.
3. Create Ignition files.
4. Set up the automated Oracle Linux installation. This includes creating a Kickstart file to use the OSTree image and Ignition configuration files.
5. Decide the order of the nodes to upgrade.

   > **Important**
   >
   > The control planes nodes must be upgraded first, then worker nodes.

   Upgrade one node at a time. For each node in the cluster:

   a. Drain the node, and reset the node.
   b. Add the node back into the cluster using the Release 2 Ignition configuration.
   c. Reboot the node using the Kickstart appropriate to the role (control plane or worker node). The node is joined to the cluster, using the Release 2 OS, as the appropriate role.
   d. Uncordon the node.
   e. Validate the node is upgraded.

Prepare the Release 2 Configuration

Set up the configuration for an Oracle CNE Release 2 cluster.

Creating a Cluster Configuration File

Create a cluster configuration file to match the configuration of the Release 1 cluster. Ensure you include any custom configuration identified in OS Customizations. The options you set must match the Release 1 cluster, for example, the cluster name must be the same.

Create a cluster configuration file. The minimum configuration required is:

```
provider: byo
name: cluster_name
kubernetesVersion: kube_version
loadBalancer: ip_address
```

```
providers:
  byo:
    networkInterface: nic_name
```

For information on what can be included in a cluster configuration file, see Oracle Cloud Native Environment: Kubernetes Clusters.

For example:

```
provider: byo
name: mycluster
kubernetesVersion: 1.29
loadBalancer: 192.0.2.100
providers:
  byo:
    networkInterface: enp1s0
```

Creating an OSTree Image

Before you begin, identify the version of Kubernetes running in the Oracle CNE Release 1 cluster.

1. Create an OSTree image.

   Important

   If you're using a custom OSTree image, created using Oracle Container Host for Kubernetes Image Builder, you don't need to perform this step.

   Create an OSTree image for the BYO provider that includes the version of Kubernetes in the Oracle CNE Release 1 cluster.

   Use the `ocne image create` command to create a OSTree image. The syntax is:

   ```
   ocne image create
   {-a|--arch} arch
   [{-t|--type} provider]
   [{-v|--version} version]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example, for Kubernetes Release 1.29 on 64-bit x86 servers:

   ```
   ocne image create --type ostree --arch amd64 --version 1.29
   ```

   This command might take some time to complete. The OSTree image is saved to the `$HOME/.ocne/images/` directory.

2. Upload the OSTree image to a container registry.

Use the `ocne image upload` command to upload the image to a container registry. The syntax is:

```
ocne image upload
{-a|--arch} arch
[{-b|--bucket} name]
[{-c|--compartment} name]
[--config path]
[{-d|--destination} path]
{-f|--file} path
{-i|--image-name} name
{-t|--type} provider
{-v|--version} version
```

For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

For example:

```
ocne image upload --type ostree --file $HOME/.ocne/images/ock-1.29-amd64-
ostree.tar --destination docker://myregistry.example.com/ock-ostree:latest
--arch amd64
```

The Kubernetes cluster uploads the OSTree image. A sign in prompt is provided if credentials aren't set for the target container registry. The image is uploaded to the container registry.

> **Tip**
>
> If you don't have a container registry, or you prefer to run this locally, you can load the OSTree archive image to a local container runtime. For example, to load an OSTree archive file to Podman on the localhost, you might use:
>
> ```
> podman load < $HOME/.ocne/images/ock-1.29-amd64-ostree.tar
> ```

3. Serve the OSTree image.

   Make the OSTree image available as a container. Use any container runtime you like, including a Kubernetes cluster.

   For example, to serve the container image from a container registry using Podman, you might use:

   ```
   podman run -d --name ock-content-server -p 8080:80 myregistry.example.com/
   ock-ostree:latest
   ```

   To serve the container image from a local instance of Podman, you could use:

   ```
   podman run -d --name ock-content-server -p 8080:80 localhost/ock-
   ostree:latest
   ```

Creating Ignition Files

In Release 2, Ignition files are needed to join a node to a Kubernetes cluster. The settings in an Ignition file differ for control plane and worker nodes, so create an Ignition file for each of the cluster node types. You include these Ignition files in the Kickstart file that's used to boot nodes when upgrading the host OS to the Release 2 OS.

1. Set up the location of Ignition files.

   Decide how you want to make the Kubernetes cluster Ignition files available.

   An Ignition file must be available to all hosts during their first boot. Ignition files can be served using any of the platforms listed in the upstream Ignition documentation, for example, using a Network File Server (NFS), or a web server.

2. Set the location of the `kubeconfig` file.

   Set the `kubeconfig` file as the `KUBECONFIG` environment variable. This must be set to the Release 1 cluster. For example:

   ```
   export KUBECONFIG=~/.kube/kubeconfig.mycluster
   ```

3. Generate the Ignition configuration for control plane nodes.

   Use the `ocne cluster join` command to generate the Ignition information that joins a control plane node to the cluster. Use the syntax:

   ```
   ocne cluster join
   [{-c|--config} path]
   [{-d|--destination} path]
   [{-N|--node} name]
   [{-P|--provider} provider]
   [{-r|--role-control-plane}]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   The important options are:

   ```
   ocne cluster join --kubeconfig path --role-control-plane --config path >
   ignition_file
   ```

   > **Important**
   >
   > The `--kubeconfig` option is required, even if this is set as an environment variable.

   For example:

   ```
   ocne cluster join --kubeconfig ~/.kube/kubeconfig.mycluster --role-control-
   plane --config myconfig.yaml > ignition-control-plane.ign
   ```

   The output includes important information used later in the upgrade.

> **Important**
>
> You don't need to run the command shown in the output. Instead, take note of the *certificate-key* and *token* in this output.

4. Generate the Ignition configuration for worker nodes.

   Use the `ocne cluster join` command to generate the Ignition information that joins a worker node to the cluster. The important options are:

   ```
   ocne cluster join --kubeconfig path --config path > ignition_file
   ```

   > **Important**
   >
   > The `--kubeconfig` option is required, even if this is set as an environment variable.

   For example:

   ```
   ocne cluster join --kubeconfig ~/.kube/kubeconfig.mycluster --config
   myconfig.yaml > ignition-worker.ign
   ```

   The output includes important information used later in the upgrade.

   > **Important**
   >
   > You don't need to run the command shown in the output. Instead, take note of the *token* in this output.

5. Expose the Ignition files.

   Make the Ignition files available at the location you chose for hosting these files. For example, copy them to a web server.

> **Tip**
>
> To make the Ignition files available locally, using an NGINX web server container running on Podman, you might use:
>
> ```
> podman run -d --name ol-content-server -p 8081:80 -v
> 'directory':/usr/share/nginx/html/ock --privileged container-
> registry.oracle.com/olcne/nginx:1.17.7
> ```
>
> Where *directory* is the location on the localhost that contains the Ignition files. This directory might also be used to serve Kickstart files. You can validate the Ignition file is being served using:
>
> ```
> curl http://IPaddress:8081/ock/filename.ign
> ```
>
> Where *IPaddress* is the IP address of the localhost, and *filename* is the name of the Ignition file.

Creating an Automated Oracle Linux Installation

Create two Kickstart files, one to start control plane nodes, and one to start worker nodes. They're highly likely to be identical, but must include references to the appropriate Ignition file for the Kubernetes node type.

> **Note**
>
> To retain existing host IP addresses, ensure you create one Kickstart file for each node and include the IP address configuration. Adjust the steps in this upgrade as appropriate.

1. Prepare the automated Oracle Linux installation.

   A Kickstart file defines an automated installation of Oracle Linux. Kickstart files must be made available during the installation of the Release 2 OS. Decide on the method to perform an automated install of Oracle Linux on the hosts using Kickstart files. For example, you might want to use a network drive, a web server, or a USB drive.

   > **Tip**
   >
   > It might be helpful to include the Kickstart files in the same location as the Ignition files, for example, using NFS, or a web server.

   You only need to provision the kernel and initrd (initial ramdisk) that matches the boot kernel. We recommend you use an Oracle Linux UEK boot ISO file as the boot media as it contains the required kernel and initrd, in a smaller file size. Download Oracle Linux ISO files from the Oracle Linux yum server.

   Prepare the Oracle Linux boot media using the method you select.

For more information about the automated installation options for Oracle Linux, see Oracle Linux 9: Installing Oracle Linux.

2. Create two Kickstart files.

   Create a Kickstart file for control plane nodes, and a separate one for worker nodes. The Kickstart files must include the OSTree image location, and the location of the Ignition file for the node type.

   Ensure you add the location of the Ignition file using:

   ```
   bootloader --append "rw ip=dhcp rd.neednet=1 ignition.platform.id=metal
   ignition.config.url=http://hostname/filename.ign ignition.firstboot=1"
   ```

   Where *hostname* is the hostname or IP address of the Ignition file server, and *filename* is the Ignition file.

   And the location of the OSTree image in the container registry using:

   ```
   ostreesetup --nogpg --osname ock --url registry --ref ock
   ```

   Replace *registry* with the URL to the OSTree image in the container registry. For example, `http://myregistry.example.com/ostree`.

   For example, the OSTree image location information might use something similar to:

   ```
   ...
   services --enabled=ostree-remount
   bootloader --append "rw ip=dhcp rd.neednet=1 ignition.platform.id=metal
   ignition.config.url=http://myhost.example.com/ignition.ign
   ignition.firstboot=1"
   ostreesetup --nogpg --osname ock --url http://myregistry.example.com/
   ostree --ref ock
   %post
   %end
   ```

   This example creates a minimal Kickstart file:

   a. Set the variables:

      ```
      export OSTREE_IP=$(ip route |  grep 'default.*src' | cut -d' ' -f9)
      export OSTREE_PORT=8080
      export OSTREE_REF=ock
      export OSTREE_PATH=ostree
      export KS_PORT=8081
      ```

   b. Create the Kickstart file:

      The following command generates a Kickstart file using the variables set in the previous step.

      ```
      envsubst > kickstart.cfg << EOF
      logging

      keyboard us
      lang en_US.UTF-8
      timezone UTC
      ```

```
text
reboot

selinux --enforcing
firewall --use-system-defaults
network --bootproto=dhcp

zerombr
clearpart --all --initlabel
part /boot --fstype=xfs --label=boot --size=2048
part /boot/efi --fstype=efi --label=efi --size=1024
part / --fstype=xfs --label=root --grow

services --enabled=ostree-remount

bootloader --append "rw ip=dhcp rd.neednet=1 ignition.platform.id=metal
ignition.config.url=http://$SERVER_IP:$KS_PORT/ock/ignition.ign
ignition.firstboot=1"

ostreesetup --nogpg --osname ock --url
http://$SERVER_IP:$OSTREE_PORT/$OSTREE_PATH --ref $OSTREE_REF

%post

%end
EOF
```

The following example Kickstart file includes static IP address configuration. A separate file is required for each host if using this option.

```
envsubst > kickstart.cfg << EOF
logging

keyboard us
lang en_US.UTF-8
timezone UTC
text
reboot

selinux --enforcing
firewall --use-system-defaults
network --bootproto=static --device=enp1s0 --gateway=192.0.2.1 --
ip=192.0.2.50 --netmask=255.255.255.0 --onboot=yes --hostname=ocne-
control-plane-1

zerombr
clearpart --all --initlabel
part /boot --fstype=xfs --label=boot --size=2048
part /boot/efi --fstype=efi --label=efi --size=1024
part / --fstype=xfs --label=root --grow

services --enabled=ostree-remount

bootloader --append "rw
ip=192.0.2.50::192.0.2.1:255.255.255.0::enp1s0:none rd.neednet=1
```

```
ignition.platform.id=metal
ignition.config.url=http://$SERVER_IP:$KS_PORT/ock/ignition.ign
ignition.firstboot=1"

ostreesetup --nogpg --osname ock --url
http://$SERVER_IP:$OSTREE_PORT/$OSTREE_PATH --ref $OSTREE_REF

%post

%end
EOF
```

3. Expose the Kickstart files.

   Make the Kickstart files available at the location you chose for hosting these files. For example, copy them to a web server.

   Tip

   You can also make the Kickstart files available locally using Podman. If you have an NGINX container running on Podman locally to serve Ignition files, and the Kickstart files are in the same directory, they're already being served. If not, start a container:

   ```
   podman run -d --name ol-content-server -p 8081:80 -v
   'directory':/usr/share/nginx/html/ock --privileged container-
   registry.oracle.com/olcne/nginx:1.17.7
   ```

   Where *directory* is the location on the localhost that contains the Kickstart files. You can validate the Kickstart file is being served using:

   ```
   curl http://IPaddress:8081/ock/filename.cfg
   ```

   Where *IPaddress* is the IP address of the localhost, and *filename* is the name of the Kickstart file.

Upgrade the Nodes

Upgrade Kubernetes nodes from Release 1 to Release 2.

For each node in the cluster, perform the following steps to upgrade from Release 1 to Release 2.

Important

The control planes nodes must be upgraded first, then worker nodes.

Preparing Nodes

Remove the Kubernetes nodes from the Release 1 cluster, and add them back using the Release 2 Ignition configuration.

1. Set the location of the `kubeconfig` file.

   Set the location of the Release 1 `kubeconfig` file as the `KUBECONFIG` environment variable. For example:

   ```
   export KUBECONFIG=~/.kube/kubeconfig.mycluster
   ```

   > **Important**
   >
   > The Kubernetes configuration file **must** be saved as the name of the Release 1 cluster (the Kubernetes Module name).

2. Find the node name.

   Find the name of the node to upgrade in the cluster.

   ```
   kubectl get nodes
   ```

3. Set the node name.

   Set the node name as an environment variable.

   ```
   export TARGET_NODE=nodename
   ```

4. Drain the node.

   Use the `kubectl drain` command to drain the node.

   For control plane nodes, use:

   ```
   kubectl drain $TARGET_NODE --ignore-daemonsets
   ```

   For worker nodes, use:

   ```
   kubectl drain $TARGET_NODE --ignore-daemonsets --delete-emptydir-data
   ```

5. Reset the node.

   Use the `ocne cluster console` command to reset the node. The syntax to use is:

   ```
   ocne cluster console
   [{-d|--direct}]
   {-N|--node} nodename
   [{-t|--toolbox}]
   [-- command]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

For example:

```
ocne cluster console --node $TARGET_NODE --direct -- kubeadm reset -f
```

6. Add the node back into the cluster.

   a. If the node is a control plane node:

      When adding a control plane node, two things need to be created, an encrypted certificate bundle, and a join token.

      Run the following command to connect to a control plane node's OS console and create the certificate bundle:

      ```
      ocne cluster console --node control_plane_name --direct -- kubeadm init
      phase upload-certs --certificate-key certificate-key --upload-certs
      ```

      Replace *control_plane_name* with a control plane node that's running in the Release 1 cluster.

      > **Important**
      >
      > This isn't the target node, but a separate control plane node that's used to run the `kubeadm init` command.

      Replace *certificate-key* with the output displayed when the Ignition information for control plane nodes was generated using the `ocne cluster join` command.

      Run the following command to create a join token:

      ```
      ocne cluster console --node control_plane_name --direct -- kubeadm
      token create token
      ```

      Replace *control_plane_name* with a control plane node that's running in the Release 1 cluster.

      Replace *token* with the output displayed when the Ignition information for control plane nodes was generated using the `ocne cluster join` command.

      > **Important**
      >
      > If the token was generated more than 24 hours prior, it has likely expired, and you must regenerate the Ignition files, which also generates a fresh token.

      Use the `kubectl get nodes` command to confirm the control plane node is added to the cluster. This might take a few moments.

      ```
      kubectl get nodes
      ```

   b. If the node is a worker node:

When adding a worker node, a join token must be created. Run the following command to connect to a control plane node's OS console to perform this step:

```
ocne cluster console --node control_plane_name --direct -- kubeadm
token create token
```

Replace *control_plane_name* with a control plane node that's running in the Release 1 cluster.

Replace *token* with the output displayed when the Ignition information for worker nodes was generated in using the `ocne cluster join` command.

> **Important**
>
> If the token was generated more than 24 hours prior, it has likely expired, and you must regenerate the Ignition files, which also generates a fresh token.

Use the `kubectl get nodes` command to confirm the worker node is added to the cluster. This might take a few moments.

```
kubectl get nodes
```

Installing the Release 2 OS

Reboot the host and reinstall the OS using the Kickstart file to install the Oracle CNE Release 2 OS image.

1. Shut down the system.
2. Boot the system.
3. Interrupt the installation.

   When the OS installation screen is displayed, select the `Install Oracle Linux release` option and press `e`. The boot options are displayed.
4. Set the Kickstart file location.

   Add the location of the Kickstart file, using the `inst.ks` option, to the boot options. For example:

   ```
   inst.ks=http://myhost.example.com/ock/control-plane.cfg
   ```

   > **Important**
   >
   > Two Kickstart files must be available, one for control plane nodes, and one for worker nodes, as they include different Ignition information. Ensure you set this to the appropriate Kickstart file.

   Enter `Ctrl` + `X` to boot the server using the Kickstart file.

   The server boots to the Release 2 image and rejoins the Kubernetes cluster.

Uncordon Nodes

Uncordon the Kubernetes nodes to enable cluster workloads to run.

1. Find the node name.

   Find the name of the node to uncordon.

   ```
   kubectl get nodes
   ```

2. Uncordon the node.

   Use the `kubectl uncordon` command to uncordon the node.

   ```
   kubectl uncordon node_name
   ```

   For example:

   ```
   kubectl uncordon ocne-control-plane-1
   ```

3. Verify the node is available.

   Use the `kubectl get nodes` command to confirm the `STATUS` column is set to `Ready`.

   ```
   kubectl get nodes
   ```

Validating the Node Upgrade

Validate a node is running the Release 2 OS.

1. List the nodes in the cluster.

   List the nodes in the Kubernetes cluster to ensure all expected nodes are listed.

   ```
   kubectl get nodes
   ```

2. Show information about the node.

   Use the `ocne cluster info` command to display information about the node. The syntax is:

   ```
   ocne cluster info
   [{-N|--nodes}] nodename, ...
   [{-s|--skip-nodes }]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne cluster info --nodes ocne-control-plane-1
   ```

3. Validate the node information.

The node is running the Release 2 image if the output of the `ocne cluster info` command looks similar to:

```
Node: ocne-control-plane-1
  Registry and tag for ostree patch images:
    registry: container-registry.oracle.com/olcne/ock-ostree
    tag: 1.29
    transport: ostree-unverified-registry
  Ostree deployments:
      ock
5d6e86d05fa0b9390c748a0a19288ca32bwer1eac42fef1c048050ce03ffb5ff9.1
(staged)
    * ock
5d6e86d05fa0b9390c748a0a19288ca32bwer1eac42fef1c048050ce03ffb5ff9.0
```

The OSTree based image information is displayed in the output.

The node isn't running the Release 2 image if the output is missing this information, and looks similar to:

```
Node: ocne-control-plane-2
  Registry and tag for ostree patch images:
  Ostree deployments:
```

5

Upgrade an Oracle Linux Virtualization Manager Cluster

Upgrade a Kubernetes cluster running on Oracle Linux Virtualization Manager. Use this to upgrade a cluster from Oracle CNE Release 1 to Release 2

To upgrade a Kubernetes cluster running on Oracle Linux Virtualization Manager, you can use the steps in Upgrade a Bring Your Own Cluster, which uses a Kickstart file to change the host OS, or the steps outlined in this section, which uses the Oracle Linux Virtualization Manager console to replace the boot disk.

The steps to upgrade a Release 1 cluster to Release 2 are:

1. Create a Release 2 cluster configuration file.
2. Create an Oracle Container Host for Kubernetes (OCK) image.
3. Create Ignition files.
4. Create configuration disks that contain the Ignition configuration.
5. Decide the order of the nodes to upgrade.

> **Important**
>
> The control planes nodes must be upgraded first, then worker nodes.

   Upgrade one node at a time. For each node in the cluster:

   a. Drain the node, and reset the node.
   b. Add the node back into the cluster using the Release 2 Ignition configuration.
   c. Replace the boot disk, add the configuration disk, and reboot the node. The node is joined to the cluster, using the Release 2 OS, as the appropriate role.
   d. Uncordon the node.
   e. Validate the node is upgraded.

Prepare the Release 2 Configuration

Set up the configuration for an Oracle CNE Release 2 cluster.

Creating Cluster Configuration Files

Create a cluster configuration file to match the configuration of the Release 1 cluster. Ensure you include any custom configuration identified in OS Customizations. The options you set must match the Release 1 cluster, for example, the cluster name must be the same.

Create a cluster configuration file for each VM. This configuration file contains the hostname and IP address information, so a configuration file must be created for each VM.

The minimum configuration required is:

```
provider: byo
name: cluster_name
kubernetesVersion: kube_version
loadBalancer: ip_address
providers:
  byo:
    networkInterface: nic_name
extraIgnitionInline: |
  variant: fcos
  version: 1.5.0
  storage:
    files:
      - path: /etc/hostname
        mode: 0755
        contents:
          inline: hostname
      - path: /etc/sysconfig/network-scripts/ifcfg-nic_name
        mode: 0755
        contents:
          inline: |
            TYPE=Ethernet
            BOOTPROTO=none
            NAME=nic_name
            DEVICE=nic_name
            ONBOOT=yes
            IPADDR=IP_address
            PREFIX=24
            GATEWAY=gateway
```

For information on what can be included in a cluster configuration file, see Oracle Cloud Native Environment: Kubernetes Clusters.

For example:

```
provider: byo
name: mycluster
kubernetesVersion: 1.29
loadBalancer: 192.0.2.100
providers:
  byo:
    networkInterface: enp1s0
extraIgnitionInline: |
  variant: fcos
  version: 1.5.0
  storage:
    files:
      - path: /etc/hostname
        mode: 0755
        contents:
          inline: ocne-control-plane-1
      - path: /etc/sysconfig/network-scripts/ifcfg-enp1s0
        mode: 0755
        contents:
          inline: |
```

```
TYPE=Ethernet
BOOTPROTO=none
NAME=enp1s0
DEVICE=enp1s0
ONBOOT=yes
IPADDR=192.0.2.50
PREFIX=24
GATEWAY=192.0.2.1
```

Creating an OCK Image

Before you begin, identify the version of Kubernetes running in the Oracle CNE Release 1 cluster.

1. Create an OCK image.

   > **Important**
   >
   > If you're using a custom OCK image, created using Oracle Container Host for Kubernetes Image Builder, you don't need to perform this step.

   Create an OCK image that includes the version of Kubernetes in the Oracle CNE Release 1 cluster.

   Use the `ocne image create` command to create an OCK image. The syntax is:

   ```
   ocne image create
   {-a|--arch} arch
   [{-t|--type} provider]
   [{-v|--version} version]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example, for Kubernetes Release 1.29 on 64-bit x86 servers:

   ```
   ocne image create --type olvm --arch amd64 --version 1.29
   ```

   This command might take some time to complete. The OCK image is saved to the `$HOME/.ocne/images/` directory.

2. Upload the OCK image to Oracle Linux Virtualization Manager.

   Use the Oracle Linux Virtualization Manager console to upload the OCK image as a disk.

3. Clone the OCK disk.

   Use the Oracle Linux Virtualization Manager console to make a clone of the OCK disk for each VM to be upgraded. There must be one disk available for each VM.

> **Important**
>
> Each VM requires a new OCK disk. An OCK disk can't be reused. It must not have been used when booting another VM, or the Ignition information isn't read (it's only read on the first boot).

Creating Ignition Files

In Release 2, Ignition information is needed to join a node to a Kubernetes cluster. An Ignition file must be generated for each VM. The settings in an Ignition file differ for control plane and worker nodes, so ensure you use the correct syntax to generate Ignition for the appropriate node type. You include the content of the Ignition file in the configuration disk that's used when booting a node during the host upgrade to the Release 2 OS.

Ensure you use the appropriate Oracle CNE cluster configuration file for the VM when using the `ocne cluster join` command to generate the Ignition file.

Repeat these steps for each VM.

1. If the node is a control plane:

   Use the `ocne cluster join` command to generate the Ignition information that joins a control plane node to the cluster. Use the syntax:

   ```
   ocne cluster join
   [{-c|--config} path]
   [{-d|--destination} path]
   [{-N|--node} name]
   [{-P|--provider} provider]
   [{-r|--role-control-plane}]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   The important options are:

   ```
   ocne cluster join --kubeconfig path --role-control-plane --config path >
   ignition_file
   ```

   > **Important**
   >
   > The `--kubeconfig` option is required, even if this is set as an environment variable.

   For example:

   ```
   ocne cluster join --kubeconfig ~/.kube/kubeconfig.mycluster --role-control-
   plane --config myconfig.yaml > ignition-control-plane.ign
   ```

   The output includes important information used later in the upgrade.

> **Important**
>
> You don't need to run the command shown in the output. Instead, take note of the *certificate-key* and *token* in this output.

2. If the node is a worker node:

   Use the `ocne cluster join` command to generate the Ignition information that joins a worker node to the cluster. The important options are:

   ```
   ocne cluster join --kubeconfig path --config path > ignition_file
   ```

   > **Important**
   >
   > The `--kubeconfig` option is required, even if this is set as an environment variable.

   For example:

   ```
   ocne cluster join --kubeconfig ~/.kube/kubeconfig.mycluster --config
   myconfig.yaml > ignition-worker.ign
   ```

   The output includes important information used later in the upgrade.

   > **Important**
   >
   > You don't need to run the command shown in the output. Instead, take note of the *token* in this output.

Creating Configuration Disks

Create a configuration disk for each VM, with the label of `CONFIG-2`, that includes the Ignition configuration for the Oracle Linux Virtualization Manager VM. The Ignition configuration is generated using the `ocne cluster join` command.

Repeat these steps for each VM, changing the Qcow2 disk name as required.

1. Load the `nbd` module.

   If this module isn't already loaded, load it using:

   ```
   sudo modprobe nbd
   ```

2. Set the target node name as an environment variable.

   ```
   export FILENAME=config2-target_node.qcow2
   ```

   Replace *target_node* with the name of the target Kubernetes node.

3. Create a configuration disk.

```
qemu-img create -f qcow2 $FILENAME 256M
```

4. Configure the disk.

```
sudo qemu-nbd --connect /dev/nbd0 $FILENAME
sudo parted -s /dev/nbd0 mklabel gpt
sudo parted -s /dev/nbd0 mkpart CONFIG-2 xfs 0 100%
sudo mkfs.xfs /dev/nbd0p1
sudo xfs_admin -L CONFIG-2 /dev/nbd0p1
```

5. Mount the disk.

```
sudo mount /dev/nbd0p1 /mnt
```

6. Make a directory for the Ignition information on the disk.

```
sudo mkdir -p /mnt/openstack/latest
```

7. Add the Ignition configuration.

   Using `sudo`, create a file named `user_data` and add the Ignition information generated in Creating Ignition Files to the file. The file to add this information is:

   `/mnt/openstack/latest/user_data`

   Add the Ignition configuration for the node.

8. Unmount the disk.

   Unmount and disconnect from the disk.

```
sudo umount /mnt
sudo qemu-nbd --disconnect /dev/nbd0
```

9. Upload the disk.

   Upload the disk to Oracle Linux Virtualization Manager using the Oracle Linux Virtualization Manager console.

Upgrade the Nodes

Upgrade Kubernetes nodes from Release 1 to Release 2.

For each node in the cluster, perform the following steps to upgrade from Release 1 to Release 2.

> **Important**
>
> The control planes nodes must be upgraded first, then worker nodes.

Preparing Nodes

Remove the Kubernetes nodes from the Release 1 cluster, and add them back using the Release 2 Ignition configuration.

1. Set the location of the `kubeconfig` file.

   Set the location of the Release 1 `kubeconfig` file as the `KUBECONFIG` environment variable. For example:

   ```
   export KUBECONFIG=~/.kube/kubeconfig.mycluster
   ```

   > **Important**
   >
   > The Kubernetes configuration file **must** be saved as the name of the Release 1 cluster (the Kubernetes Module name).

2. Find the node name.

   Find the name of the node to upgrade in the cluster.

   ```
   kubectl get nodes
   ```

3. Set the node name.

   Set the node name as an environment variable.

   ```
   export TARGET_NODE=nodename
   ```

4. Drain the node.

   Use the `kubectl drain` command to drain the node.

   For control plane nodes, use:

   ```
   kubectl drain $TARGET_NODE --ignore-daemonsets
   ```

   For worker nodes, use:

   ```
   kubectl drain $TARGET_NODE --ignore-daemonsets --delete-emptydir-data
   ```

5. Reset the node.

   Use the `ocne cluster console` command to reset the node. The syntax to use is:

   ```
   ocne cluster console
   [{-d|--direct}]
   {-N|--node} nodename
   [{-t|--toolbox}]
   [-- command]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

For example:

```
ocne cluster console --node $TARGET_NODE --direct -- kubeadm reset -f
```

6. Add the node back into the cluster.

   a. If the node is a control plane node:

      When adding a control plane node, two things need to be created, an encrypted certificate bundle, and a join token.

      Run the following command to connect to a control plane node's OS console and create the certificate bundle:

      ```
      ocne cluster console --node control_plane_name --direct -- kubeadm init
      phase upload-certs --certificate-key certificate-key --upload-certs
      ```

      Replace *control_plane_name* with a control plane node that's running in the Release 1 cluster.

      > **Important**
      >
      > This isn't the target node, but a separate control plane node that's used to run the `kubeadm init` command.

      Replace *certificate-key* with the output displayed when the Ignition information for control plane nodes was generated using the `ocne cluster join` command.

      Run the following command to create a join token:

      ```
      ocne cluster console --node control_plane_name --direct -- kubeadm
      token create token
      ```

      Replace *control_plane_name* with a control plane node that's running in the Release 1 cluster.

      Replace *token* with the output displayed when the Ignition information for control plane nodes was generated using the `ocne cluster join` command.

      > **Important**
      >
      > If the token was generated more than 24 hours prior, it has likely expired, and you must regenerate the Ignition files, which also generates a fresh token.

      Use the `kubectl get nodes` command to confirm the control plane node is added to the cluster. This might take a few moments.

      ```
      kubectl get nodes
      ```

   b. If the node is a worker node:

When adding a worker node, a join token must be created. Run the following command to connect to a control plane node's OS console to perform this step:

```
ocne cluster console --node control_plane_name --direct -- kubeadm
token create token
```

Replace *control_plane_name* with a control plane node that's running in the Release 1 cluster.

Replace *token* with the output displayed when the Ignition information for worker nodes was generated in using the `ocne cluster join` command.

> **Important**
>
> If the token was generated more than 24 hours prior, it has likely expired, and you must regenerate the Ignition files, which also generates a fresh token.

Use the `kubectl get nodes` command to confirm the worker node is added to the cluster. This might take a few moments.

```
kubectl get nodes
```

Replacing the Disks

Replace the Oracle Linux Virtualization Manager boot disk and add the configuration disk to a VM, and reboot it to install the Release 2 image. The node is then running the Release 2 image and added back to the Kubernetes cluster.

1. Sign in to the Oracle Linux Virtualization Manager console.
2. Shut down the VM.
3. Remove the existing boot disk.
4. Attach the OCK disk.

   Attach one of the cloned OCK image disks to the VM. Ensure you set the `bootable=true` option.
5. Attach the configuration disk.

   Attach the appropriate configuration disk, with the label `CONFIG-2`, created in Creating Configuration Disks. Ensure you attach the configuration disk that includes the Ignition information appropriate for the node.
6. Boot the VM.

   Start up the VM using the new OCK disk, and the configuration disk that contains the Ignition configuration.

   The VM boots using the new disks. The VM is joined into the cluster using Ignition information contained in the configuration disk.

Uncordon Nodes

Uncordon the Kubernetes nodes to enable cluster workloads to run.

1. Find the node name.

   Find the name of the node to uncordon.

   ```
   kubectl get nodes
   ```

2. Uncordon the node.

   Use the `kubectl uncordon` command to uncordon the node.

   ```
   kubectl uncordon node_name
   ```

   For example:

   ```
   kubectl uncordon ocne-control-plane-1
   ```

3. Verify the node is available.

   Use the `kubectl get nodes` command to confirm the `STATUS` column is set to `Ready`.

   ```
   kubectl get nodes
   ```

Validating the Node Upgrade

Validate a node is running the Release 2 OS.

1. List the nodes in the cluster.

   List the nodes in the Kubernetes cluster to ensure all expected nodes are listed.

   ```
   kubectl get nodes
   ```

2. Show information about the node.

   Use the `ocne cluster info` command to display information about the node. The syntax is:

   ```
   ocne cluster info
   [{-N|--nodes}] nodename, ...
   [{-s|--skip-nodes }]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne cluster info --nodes ocne-control-plane-1
   ```

3. Validate the node information.

   The node is running the Release 2 image if the output of the `ocne cluster info` command looks similar to:

   ```
   Node: ocne-control-plane-1
     Registry and tag for ostree patch images:
       registry: container-registry.oracle.com/olcne/ock-ostree
       tag: 1.29
   ```

```
    transport: ostree-unverified-registry
  Ostree deployments:
     ock
5d6e86d05fa0b9390c748a0a19288ca32bwer1eac42fef1c048050ce03ffb5ff9.1
(staged)
    * ock
5d6e86d05fa0b9390c748a0a19288ca32bwer1eac42fef1c048050ce03ffb5ff9.0
```

The OSTree based image information is displayed in the output.

The node isn't running the Release 2 image if the output is missing this information, and looks similar to:

```
Node: ocne-control-plane-2
  Registry and tag for ostree patch images:
  Ostree deployments:
```

6

Upgrade an OCI Cluster

Upgrade a cluster running on Oracle Cloud Infrastructure (OCI). Use this to upgrade a cluster from Oracle CNE Release 1 to Release 2

The steps to upgrade a Release 1 cluster to Release 2 are:

1. Set up OCI.
2. Create a Release 2 cluster configuration file.
3. Create an Oracle Container Host for Kubernetes (OCK) image.
4. Create Ignition files.
5. Decide the order of the nodes to upgrade.

   > **Important**
   >
   > The control planes nodes must be upgraded first, then worker nodes.

   Upgrade one node at a time. For each node in the cluster:

   a. Drain the node, and reset the node.
   b. Add the node back into the cluster using the Release 2 Ignition configuration.
   c. Replace the boot volume and reboot the node. The node is joined to the cluster, using the Release 2 OS, as the appropriate role.
   d. Uncordon the node.
   e. Validate the node is upgraded.

Prepare the Release 2 Configuration

Set up the configuration for an Oracle CNE Release 2 cluster.

Setting Up OCI

Set up OCI to perform an upgrade.

Create an Object Storage bucket, and gather OCIDs.

1. Create an Object Storage bucket.

   Create an Object Storage bucket in OCI. The default bucket name is `ocne-images`. If you use a different name, ensure you set this in the cluster configuration file.

   For information on creating an Object Storage bucket, see the Oracle Cloud Infrastructure documentation.

2. Identify OCIDs.

   Identify OCIDs for the following resources:

- The compartment in which to deploy resources.
- The Object Storage bucket.

Creating a Cluster Configuration File

Create a cluster configuration file to match the configuration of the Release 1 cluster. Ensure you include any custom configuration identified in OS Customizations. The options you set must match the Release 1 cluster, for example, the cluster name must be the same.

1. Create a cluster configuration file.

   The minimum configuration required is:

   ```
   provider: byo
   name: cluster_name
   kubernetesVersion: kube_version
   loadBalancer: ip_address
   providers:
     byo:
       networkInterface: nic_name
   ```

   For information on what can be included in a cluster configuration file, see Oracle Cloud Native Environment: Kubernetes Clusters.

   For example:

   ```
   provider: byo
   name: mycluster
   kubernetesVersion: 1.29
   loadBalancer: 192.0.2.100
   providers:
     byo:
       networkInterface: enp1s0
   ```

2. Add extra configuration.

   If the source cluster includes the OCI Cloud Controller Manager Module, include the following in the cluster configuration file.

   ```
   extraIgnitionInline: |
     variant: fcos
     version: 1.5.0
     systemd:
       units:
         - name: iscsid.service
           enabled: true
   ```

Creating an OCK Image

Before you begin, identify the version of Kubernetes running in the Oracle CNE Release 1 cluster. Also identify the compute instance architecture type.

1. Create an OCK image.

   Create an OCK image that includes the version of Kubernetes in the Oracle CNE Release 1 cluster.

Use the `ocne image create` command to create an OCK image. The syntax is:

```
ocne image create
{-a|--arch} arch
[{-t|--type} provider]
[{-v|--version} version]
```

For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

For example, for Kubernetes Release 1.29 on 64-bit x86 servers:

```
ocne image create --type oci --arch amd64 --version 1.29
```

This command might take some time to complete. The OCK image is saved to the `$HOME/.ocne/images/` directory.

2. Upload the OCK image to OCI.

   Use the `ocne image upload` command to upload the image to OCI. The syntax is:

   ```
   ocne image upload
   {-a|--arch} arch
   [{-b|--bucket} name]
   [{-c|--compartment} name]
   [--config path]
   [{-d|--destination} path]
   {-f|--file} path
   {-i|--image-name} name
   {-t|--type} provider
   {-v|--version} version
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne image upload --type oci --compartment compartment_OCID --bucket
   bucket_OCID --file $HOME/.ocne/images/boot.qcow2-1.29-amd64.oci --image-
   name ocne129 --arch amd64 --version 1.29
   ```

   The image is uploaded to the Object Bucket store, and then automatically converted to a custom compute image.

   Tip

   Sign in to the OCI console to monitor the upload of the image to the Object Bucket store, and the creation of a custom compute image.

Creating Ignition Files

In Release 2, Ignition files are needed to join a node to a Kubernetes cluster. The settings in an Ignition file differ for control plane and worker nodes, so create an Ignition file for each of the types. You include the content of the Ignition files when you replace the boot volume for the OCI instance.

1. Set the location of the `kubeconfig` file.

   Set the `kubeconfig` file as the `KUBECONFIG` environment variable. This must be set to the Release 1 cluster. For example:

   ```
   export KUBECONFIG=~/.kube/kubeconfig.mycluster
   ```

2. Generate the Ignition configuration for control plane nodes.

   Use the `ocne cluster join` command to generate the Ignition information that joins a control plane node to the cluster. Use the syntax:

   ```
   ocne cluster join
   [{-c|--config} path]
   [{-d|--destination} path]
   [{-N|--node} name]
   [{-P|--provider} provider]
   [{-r|--role-control-plane}]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   The important options are:

   ```
   ocne cluster join --kubeconfig path --role-control-plane --config path >
   ignition_file
   ```

   > **Important**
   >
   > The `--kubeconfig` option is required, even if this is set as an environment variable.

   For example:

   ```
   ocne cluster join --kubeconfig ~/.kube/kubeconfig.mycluster --role-control-
   plane --config myconfig.yaml > ignition-control-plane.ign
   ```

   The output includes important information used later in the upgrade.

   > **Important**
   >
   > You don't need to run the command shown in the output. Instead, take note of the *certificate-key* and *token* in this output.

3. Encode the Ignition file.

   The Ignition information must be base64 encoded to load it into OCI. For example:

   ```
   openssl enc -base64 -in ignition-control-plane.ign -out ignition-control-
   plane-base64.ign
   ```

4. Generate the Ignition configuration for worker nodes.

Use the `ocne cluster join` command to generate the Ignition information that joins a worker node to the cluster. The important options are:

```
ocne cluster join --kubeconfig path --config path > ignition_file
```

> **Important**
>
> The `--kubeconfig` option is required, even if this is set as an environment variable.

For example:

```
ocne cluster join --kubeconfig ~/.kube/kubeconfig.mycluster --config
myconfig.yaml > ignition-worker.ign
```

The output includes important information used later in the upgrade.

> **Important**
>
> You don't need to run the command shown in the output. Instead, take note of the *token* in this output.

5. Encode the Ignition file.

   For example:

   ```
   openssl enc -base64 -in ignition-worker.ign -out ignition-worker-base64.ign
   ```

Upgrade the Nodes

Upgrade Kubernetes nodes from Release 1 to Release 2.

For each node in the cluster, perform the following steps to upgrade from Release 1 to Release 2.

> **Important**
>
> The control planes nodes must be upgraded first, then worker nodes.

Preparing Nodes

Remove the Kubernetes nodes from the cluster, and add them back using the Release 2 Ignition configuration.

1. Set the location of the `kubeconfig` file.

Set the location of the Release 1 `kubeconfig` file as the `KUBECONFIG` environment variable. For example:

```
export KUBECONFIG=~/.kube/kubeconfig.mycluster
```

> **Important**
>
> The Kubernetes configuration file **must** be saved as the name of the Release 1 cluster (the Kubernetes Module name).

2. Find the node name.

   Find the name of the node to upgrade in the cluster.

   ```
   kubectl get nodes
   ```

3. Set the node name.

   Set the node name as an environment variable.

   ```
   export TARGET_NODE=nodename
   ```

4. Drain the node.

   Use the `kubectl drain` command to drain the node.

   For control plane nodes, use:

   ```
   kubectl drain $TARGET_NODE --ignore-daemonsets
   ```

   For worker nodes, use:

   ```
   kubectl drain $TARGET_NODE --ignore-daemonsets --delete-emptydir-data
   ```

5. Reset the node.

   Use the `ocne cluster console` command to reset the node. The syntax to use is:

   ```
   ocne cluster console
   [{-d|--direct}]
   {-N|--node} nodename
   [{-t|--toolbox}]
   [-- command]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne cluster console --node $TARGET_NODE --direct -- kubeadm reset -f
   ```

6. Add the node back into the cluster.

   a. If the node is a control plane node:

      When adding a control plane node, two things need to be created, an encrypted certificate bundle, and a join token.

Run the following command to connect to a control plane node's OS console and create the certificate bundle:

```
ocne cluster console --node control_plane_name --direct -- kubeadm init
phase upload-certs --certificate-key certificate-key --upload-certs
```

Replace *control_plane_name* with a control plane node that's running in the Release 1 cluster.

> **Important**
>
> This isn't the target node, but a separate control plane node that's used to run the `kubeadm init` command.

Replace *certificate-key* with the output displayed when the Ignition information for control plane nodes was generated using the `ocne cluster join` command.

Run the following command to create a join token:

```
ocne cluster console --node control_plane_name --direct -- kubeadm
token create token
```

Replace *control_plane_name* with a control plane node that's running in the Release 1 cluster.

Replace *token* with the output displayed when the Ignition information for control plane nodes was generated using the `ocne cluster join` command.

> **Important**
>
> If the token was generated more than 24 hours prior, it has likely expired, and you must regenerate the Ignition files, which also generates a fresh token.

Use the `kubectl get nodes` command to confirm the control plane node is added to the cluster. This might take a few moments.

```
kubectl get nodes
```

b. If the node is a worker node:

When adding a worker node, a join token must be created. Run the following command to connect to a control plane node's OS console to perform this step:

```
ocne cluster console --node control_plane_name --direct -- kubeadm
token create token
```

Replace *control_plane_name* with a control plane node that's running in the Release 1 cluster.

Replace *token* with the output displayed when the Ignition information for worker nodes was generated in using the `ocne cluster join` command.

> **Important**
>
> If the token was generated more than 24 hours prior, it has likely expired, and you must regenerate the Ignition files, which also generates a fresh token.

Use the `kubectl get nodes` command to confirm the worker node is added to the cluster. This might take a few moments.

```
kubectl get nodes
```

Replacing the Boot Volume

Replace the boot volume on the instance with the custom image created when the OCK image for Release 2 was loaded. This completes the upgrade of the Kubernetes node.

For more information about replacing a boot volume, see the OCI documentation.

1. Navigate to the **Replace Boot Volume** page.

   Sign in to the OCI console, and navigate to the **Replace Boot Volume** page for the instance.

2. Replace the boot volume.

   Select the **Image** option in the **Replace by** field.

   Select the **Select from a list** option in the **Apply boot volume by** field.

   Select the custom image from the **Select image** drop down.

   Set the **Boot volume size (GB)** to be at least 50GB.

   Select the **Show advanced options** section, then select **Metadata**.

   Enter `user_data` in the **Name** field, and paste in the base64 encoded Ignition information in the **Value** field. This is the content of the base64 Ignition file for the appropriate node type, either a control plane or a worker node. This was generated in Creating Ignition Files.

   Click **Replace**.

3. Reboot the instance.

   If the instance is running, it shuts down, and reboots using the new boot volume. If the instance is stopped, start the instance to boot it using the new boot volume.

   The instance boots using the new boot volume. The instance is joined into the cluster using Ignition information contained in the boot volume.

Uncordon Nodes

Uncordon the Kubernetes nodes to enable cluster workloads to run.

1. Find the node name.

   Find the name of the node to uncordon.

   ```
   kubectl get nodes
   ```

2. Uncordon the node.

Use the `kubectl uncordon` command to uncordon the node.

```
kubectl uncordon node_name
```

For example:

```
kubectl uncordon ocne-control-plane-1
```

3. Verify the node is available.

   Use the `kubectl get nodes` command to confirm the `STATUS` column is set to `Ready`.

   ```
   kubectl get nodes
   ```

Validating the Node Upgrade

Validate a node is running the Release 2 OS.

1. List the nodes in the cluster.

   List the nodes in the Kubernetes cluster to ensure all expected nodes are listed.

   ```
   kubectl get nodes
   ```

2. Show information about the node.

   Use the `ocne cluster info` command to display information about the node. The syntax is:

   ```
   ocne cluster info
   [{-N|--nodes}] nodename, ...
   [{-s|--skip-nodes }]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne cluster info --nodes ocne-control-plane-1
   ```

3. Validate the node information.

   The node is running the Release 2 image if the output of the `ocne cluster info` command looks similar to:

   ```
   Node: ocne-control-plane-1
     Registry and tag for ostree patch images:
       registry: container-registry.oracle.com/olcne/ock-ostree
       tag: 1.29
       transport: ostree-unverified-registry
     Ostree deployments:
         ock
   5d6e86d05fa0b9390c748a0a19288ca32bwer1eac42fef1c048050ce03ffb5ff9.1
   (staged)
       * ock
   5d6e86d05fa0b9390c748a0a19288ca32bwer1eac42fef1c048050ce03ffb5ff9.0
   ```

The OSTree based image information is displayed in the output.

The node isn't running the Release 2 image if the output is missing this information, and looks similar to:

```
Node: ocne-control-plane-2
  Registry and tag for ostree patch images:
  Ostree deployments:
```

7

Validate the Upgrade

Validate the upgrade from Oracle CNE Release 1 to Release 2 succeeded, and that all nodes in the Kubernetes cluster are running the Release 2 OS.

1. List the nodes in the cluster.

   List the nodes in the Kubernetes cluster to ensure all expected nodes are listed.

   ```
   kubectl get nodes
   ```

   Confirm each node `STATUS` is listed as `Ready`.

2. Show information about the cluster.

   Use the `ocne cluster info` command to display information about the cluster. The syntax is:

   ```
   ocne cluster info
   [{-N|--nodes}] nodename, ...
   [{-s|--skip-nodes }]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne cluster info
   ```

3. Validate the cluster information.

   All nodes in the cluster are running the Release 2 image if the output of the `ocne cluster info` command looks similar to:

   ```
   ...
   Node: ocne-control-plane-1
     Registry and tag for ostree patch images:
       registry: container-registry.oracle.com/olcne/ock-ostree
       tag: 1.29
       transport: ostree-unverified-registry
     Ostree deployments:
         ock
   5d6e86d05fa0b9390c748a0a19288ca32bwer1eac42fef1c048050ce03ffb5ff9.1
   (staged)
       * ock
   5d6e86d05fa0b9390c748a0a19288ca32bwer1eac42fef1c048050ce03ffb5ff9.0

   Node: ocne-control-plane-2
     Registry and tag for ostree patch images:
       registry: container-registry.oracle.com/olcne/ock-ostree
       tag: 1.29
       transport: ostree-unverified-registry
     Ostree deployments:
   ```

```
      ock
5d6e86d05fa0b9390c748a0a19288ca32bwer1eac42fef1c048050ce03ffb5ff9.1
(staged)
    * ock
5d6e86d05fa0b9390c748a0a19288ca32bwer1eac42fef1c048050ce03ffb5ff9.0
...
```

If any of the nodes aren't running the Release 2 image, the output doesn't include the registry and OSTree information, for example:

```
...
Node: ocne-control-plane-1
  Registry and tag for ostree patch images:
    registry: container-registry.oracle.com/olcne/ock-ostree
    tag: 1.29
    transport: ostree-unverified-registry
  Ostree deployments:
      ock
5d6e86d05fa0b9390c748a0a19288ca32bwer1eac42fef1c048050ce03ffb5ff9.1
(staged)
    * ock
5d6e86d05fa0b9390c748a0a19288ca32bwer1eac42fef1c048050ce03ffb5ff9.0

Node: ocne-control-plane-2
  Registry and tag for ostree patch images:
  Ostree deployments:
...
```

4. Verify the list of available applications.

   Use the `ocne catalog search` command to see the list of applications available in the Oracle catalog. The syntax is:

   ```
   ocne catalog search
   [{-N|--name} name]
   [{-p|--pattern} pattern]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example, to list all the applications available in the Oracle catalog:

   ```
   ocne catalog search
   ```

   Ensure all expected applications are available. Verify all Release 1 modules are listed as applications.

5. Verify the list of installed applications.

   Use the `ocne application list` command to see the list of applications installed in the cluster. The syntax is:

   ```
   ocne application {list|ls}
   [{-A|--all}]
   [{-n|--namespace} namespace]
   ```

For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

For example, to list all the applications installed in the cluster:

```
ocne application list --all
```

Ensure all expected installed applications are listed.

8

Migrate Other Release 1 Components

Upgrade Release 1 components and modules that aren't managed by Helm.

All Oracle CNE Release 2 applications in the application catalog are managed by Helm. During the cluster upgrade, all installed applications managed by Helm are automatically added to the application catalog. This means they can be upgraded to the latest release using the CLI or UI, when required. Upgrading applications isn't part of the upgrade from Release 1 to Release 2, and can be done at any time after the formal upgrade in this document. For information on upgrading applications in Release 2, see Oracle Cloud Native Environment: Applications.

Some Release 1 applications need manual intervention to change them so they're managed by Helm, and then the application must be manually upgraded. If any of these modules or components are in the Release 1 cluster, follow the steps in this section to update them:

- Flannel
- Istio Module

Upgrading Flannel

Upgrade Flannel in the Kubernetes cluster by removing all the Flannel components and installing the Flannel application from the application catalog.

1. Check the Flannel version.

   ```
   kubectl describe pod --namespace kube-system kube-flannel
   ```

   Enter the **Tab** key to auto complete the name of the `kube-flannel` pod.

   The output contains the `Image` option, which shows the Flannel version, for example:

   ```
   Image: container-registry.oracle.com/olcne/flannel:v0.14.1-4
   ```

2. Delete the Flannel DaemonSet.

   ```
   kubectl delete daemonsets kube-flannel-ds --namespace kube-system
   ```

3. Delete the Flannel ServiceAccount.

   ```
   kubectl delete serviceaccounts flannel --namespace kube-system
   ```

4. Delete the Flannel ConfigMap.

   ```
   kubectl delete configmap kube-flannel-cfg --namespace kube-system
   ```

5. Delete the Flannel ClusterRole.

   ```
   kubectl delete clusterrole flannel --namespace kube-system
   ```

6. Delete the Flannel ClusterRoleBinding.

   ```
   kubectl delete clusterrolebinding flannel --namespace kube-system
   ```

7. Install Flannel from the application catalog.

   ```
   ocne application install --name flannel --release flannel --namespace kube-
   flannel
   ```

8. Validate the Flannel installation.

   ```
   kubectl describe pod --namespace kube-flannel kube-flannel
   ```

   Enter the **Tab** key to auto complete the name of the `kube-flannel` pod.

   The output contains the `Image` option, which shows the Flannel version, is now `flannel:current`. For example:

   ```
   Image: container-registry.oracle.com/olcne/flannel:current
   ```

Upgrading Istio

Upgrade Istio in the Kubernetes cluster by changing all the Istio components so they're managed by Helm, and installing the latest version of Istio from the application catalog.

This upgrades Istio Release 1.20.4 to Release 1.20.5. Release 1.20.4 is the latest available Istio version available with Oracle CNE Release 1.9. We recommend you upgrade Istio to the latest available version listed in the application catalog, which might be a newer release than shown here.

1. Check the Istio version.

   ```
   kubectl describe pod --namespace istio-system istiod
   ```

   Enter the **Tab** key to auto complete the name of the `istiod` pod.

   The output contains the `Image` option, which shows the Istio version, for example:

   ```
    Image: container-registry.oracle.com/olcne/pilot:1.20.4
   ```

2. Search the application catalog for Istio versions.

   Several Istio versions are available to install. We recommend you install the latest version. To see the list of available versions, use:

   ```
   ocne catalog search --pattern istio
   ```

3. Change the Istio objects to be managed by Helm.

   Change the Istio Base objects:

   ```
   kubectl --namespace istio-system label ServiceAccount istio-reader-service-
   account app.kubernetes.io/managed-by=Helm
   kubectl --namespace istio-system annotate ServiceAccount istio-reader-
   service-account meta.helm.sh/release-name=istio-base
   ```

```
kubectl --namespace istio-system annotate ServiceAccount istio-reader-
service-account meta.helm.sh/release-namespace=istio-system

kubectl label ValidatingWebhookConfiguration istiod-default-validator
app.kubernetes.io/managed-by=Helm
kubectl annotate ValidatingWebhookConfiguration istiod-default-validator
meta.helm.sh/release-name=istio-base
kubectl annotate ValidatingWebhookConfiguration istiod-default-validator
meta.helm.sh/release-namespace=istio-system
```

Change the Istiod objects:

```
kubectl --namespace istio-system label ServiceAccount istiod
app.kubernetes.io/managed-by=Helm
kubectl --namespace istio-system annotate ServiceAccount istiod
meta.helm.sh/release-name=istiod
kubectl --namespace istio-system annotate ServiceAccount istiod
meta.helm.sh/release-namespace=istio-system

kubectl --namespace istio-system label ConfigMap istio app.kubernetes.io/
managed-by=Helm
kubectl --namespace istio-system annotate ConfigMap istio meta.helm.sh/
release-name=istiod
kubectl --namespace istio-system annotate ConfigMap istio meta.helm.sh/
release-namespace=istio-system

kubectl --namespace istio-system label ConfigMap istio-sidecar-injector
app.kubernetes.io/managed-by=Helm
kubectl --namespace istio-system annotate ConfigMap istio-sidecar-injector
meta.helm.sh/release-name=istiod
kubectl --namespace istio-system annotate ConfigMap istio-sidecar-injector
meta.helm.sh/release-namespace=istio-system

kubectl label ClusterRole istiod-clusterrole-istio-system
app.kubernetes.io/managed-by=Helm
kubectl annotate ClusterRole istiod-clusterrole-istio-system meta.helm.sh/
release-name=istiod
kubectl annotate ClusterRole istiod-clusterrole-istio-system meta.helm.sh/
release-namespace=istio-system

kubectl label ClusterRole istiod-gateway-controller-istio-system
app.kubernetes.io/managed-by=Helm
kubectl annotate ClusterRole istiod-gateway-controller-istio-system
meta.helm.sh/release-name=istiod
kubectl annotate ClusterRole istiod-gateway-controller-istio-system
meta.helm.sh/release-namespace=istio-system

kubectl label ClusterRole istio-reader-clusterrole-istio-system
app.kubernetes.io/managed-by=Helm
kubectl annotate ClusterRole istio-reader-clusterrole-istio-system
meta.helm.sh/release-name=istiod
kubectl annotate ClusterRole istio-reader-clusterrole-istio-system
meta.helm.sh/release-namespace=istio-system

kubectl label ClusterRoleBinding istiod-clusterrole-istio-system
```

```
app.kubernetes.io/managed-by=Helm
kubectl annotate ClusterRoleBinding istiod-clusterrole-istio-system
meta.helm.sh/release-name=istiod
kubectl annotate ClusterRoleBinding istiod-clusterrole-istio-system
meta.helm.sh/release-namespace=istio-system

kubectl label ClusterRoleBinding istiod-gateway-controller-istio-system
app.kubernetes.io/managed-by=Helm
kubectl annotate ClusterRoleBinding istiod-gateway-controller-istio-system
meta.helm.sh/release-name=istiod
kubectl annotate ClusterRoleBinding istiod-gateway-controller-istio-system
meta.helm.sh/release-namespace=istio-system

kubectl label ClusterRoleBinding istio-reader-clusterrole-istio-system
app.kubernetes.io/managed-by=Helm
kubectl annotate ClusterRoleBinding istio-reader-clusterrole-istio-system
meta.helm.sh/release-name=istiod
kubectl annotate ClusterRoleBinding istio-reader-clusterrole-istio-system
meta.helm.sh/release-namespace=istio-system

kubectl --namespace istio-system label Role istiod app.kubernetes.io/
managed-by=Helm
kubectl --namespace istio-system annotate Role istiod meta.helm.sh/release-
name=istiod
kubectl --namespace istio-system annotate Role istiod meta.helm.sh/release-
namespace=istio-system

kubectl --namespace istio-system label RoleBinding istiod
app.kubernetes.io/managed-by=Helm
kubectl --namespace istio-system annotate RoleBinding istiod meta.helm.sh/
release-name=istiod
kubectl --namespace istio-system annotate RoleBinding istiod meta.helm.sh/
release-namespace=istio-system

kubectl --namespace istio-system label Service istiod app.kubernetes.io/
managed-by=Helm
kubectl --namespace istio-system annotate Service istiod meta.helm.sh/
release-name=istiod
kubectl --namespace istio-system annotate Service istiod meta.helm.sh/
release-namespace=istio-system

kubectl --namespace istio-system label Deployment istiod app.kubernetes.io/
managed-by=Helm
kubectl --namespace istio-system annotate Deployment istiod meta.helm.sh/
release-name=istiod
kubectl --namespace istio-system annotate Deployment istiod meta.helm.sh/
release-namespace=istio-system

kubectl label MutatingWebhookConfiguration istio-sidecar-injector
app.kubernetes.io/managed-by=Helm
kubectl annotate MutatingWebhookConfiguration istio-sidecar-injector
meta.helm.sh/release-name=istiod
kubectl annotate MutatingWebhookConfiguration istio-sidecar-injector
meta.helm.sh/release-namespace=istio-system

kubectl --namespace istio-system label HorizontalPodAutoscaler istiod
```

```
app.kubernetes.io/managed-by=Helm
kubectl --namespace istio-system annotate HorizontalPodAutoscaler istiod
meta.helm.sh/release-name=istiod
kubectl --namespace istio-system annotate HorizontalPodAutoscaler istiod
meta.helm.sh/release-namespace=istio-system
```

Change the Istio-Ingress Gateway objects:

```
kubectl --namespace istio-system label ServiceAccount istio-ingressgateway-
service-account app.kubernetes.io/managed-by=Helm
kubectl --namespace istio-system annotate ServiceAccount istio-
ingressgateway-service-account meta.helm.sh/release-name=istio-
ingressgateway
kubectl --namespace istio-system annotate ServiceAccount istio-
ingressgateway-service-account meta.helm.sh/release-namespace=istio-system

kubectl --namespace istio-system label Service istio-ingressgateway
app.kubernetes.io/managed-by=Helm
kubectl --namespace istio-system annotate Service istio-ingressgateway
meta.helm.sh/release-name=istio-ingressgateway
kubectl --namespace istio-system annotate Service istio-ingressgateway
meta.helm.sh/release-namespace=istio-system

kubectl --namespace istio-system label Deployment istio-ingressgateway
app.kubernetes.io/managed-by=Helm
kubectl --namespace istio-system annotate Deployment istio-ingressgateway
meta.helm.sh/release-name=istio-ingressgateway
kubectl --namespace istio-system annotate Deployment istio-ingressgateway
meta.helm.sh/release-namespace=istio-system

kubectl --namespace istio-system label Role istio-ingressgateway-sds
app.kubernetes.io/managed-by=Helm
kubectl --namespace istio-system annotate Role istio-ingressgateway-sds
meta.helm.sh/release-name=istio-ingressgateway
kubectl --namespace istio-system annotate Role istio-ingressgateway-sds
meta.helm.sh/release-namespace=istio-system

kubectl --namespace istio-system label RoleBinding istio-ingressgateway-
sds app.kubernetes.io/managed-by=Helm
kubectl --namespace istio-system annotate RoleBinding istio-ingressgateway-
sds meta.helm.sh/release-name=istio-ingressgateway
kubectl --namespace istio-system annotate RoleBinding istio-ingressgateway-
sds meta.helm.sh/release-namespace=istio-system

kubectl --namespace istio-system label HorizontalPodAutoscaler istio-
ingressgateway app.kubernetes.io/managed-by=Helm
kubectl --namespace istio-system annotate HorizontalPodAutoscaler istio-
ingressgateway meta.helm.sh/release-name=istio-ingressgateway
kubectl --namespace istio-system annotate HorizontalPodAutoscaler istio-
ingressgateway meta.helm.sh/release-namespace=istio-system
```

Change the Istio-Egress Gateway objects:

```
kubectl --namespace istio-system label ServiceAccount istio-egressgateway-
service-account app.kubernetes.io/managed-by=Helm
```

```
kubectl --namespace istio-system annotate ServiceAccount istio-
egressgateway-service-account meta.helm.sh/release-name=istio-egressgateway
kubectl --namespace istio-system annotate ServiceAccount istio-
egressgateway-service-account meta.helm.sh/release-namespace=istio-system

kubectl --namespace istio-system label Service istio-egressgateway
app.kubernetes.io/managed-by=Helm
kubectl --namespace istio-system annotate Service istio-egressgateway
meta.helm.sh/release-name=istio-egressgateway
kubectl --namespace istio-system annotate Service istio-egressgateway
meta.helm.sh/release-namespace=istio-system

kubectl --namespace istio-system label Deployment istio-egressgateway
app.kubernetes.io/managed-by=Helm
kubectl --namespace istio-system annotate Deployment istio-egressgateway
meta.helm.sh/release-name=istio-egressgateway
kubectl --namespace istio-system annotate Deployment istio-egressgateway
meta.helm.sh/release-namespace=istio-system

kubectl --namespace istio-system label Role istio-egressgateway-sds
app.kubernetes.io/managed-by=Helm
kubectl --namespace istio-system annotate Role istio-egressgateway-sds
meta.helm.sh/release-name=istio-egressgateway
kubectl --namespace istio-system annotate Role istio-egressgateway-sds
meta.helm.sh/release-namespace=istio-system

kubectl --namespace istio-system label RoleBinding istio-egressgateway-sds
app.kubernetes.io/managed-by=Helm
kubectl --namespace istio-system annotate RoleBinding istio-egressgateway-
sds meta.helm.sh/release-name=istio-egressgateway
kubectl --namespace istio-system annotate RoleBinding istio-egressgateway-
sds meta.helm.sh/release-namespace=istio-system

kubectl --namespace istio-system label HorizontalPodAutoscaler istio-
egressgateway app.kubernetes.io/managed-by=Helm
kubectl --namespace istio-system annotate HorizontalPodAutoscaler istio-
egressgateway meta.helm.sh/release-name=istio-egressgateway
kubectl --namespace istio-system annotate HorizontalPodAutoscaler istio-
egressgateway meta.helm.sh/release-namespace=istio-system
```

4. Install Istio from the application catalog.

   Ensure you install the latest version of Istio. This might be later than the one shown in this example.

```
ocne application install --namespace istio-system --name istio-base --
release istio-base --version 1.20.5
ocne application install --namespace istio-system --name istiod --release
istiod --version 1.20.5
ocne application install --namespace istio-system --name istio-ingress --
release istio-ingressgateway --version 1.20.5
ocne application install --namespace istio-system --name istio-egress --
release istio-egressgateway --version 1.20.5
```

5. Restart the Istio pods.

```
kubectl rollout restart deployment --namespace istio-system istiod
kubectl rollout restart deployment --namespace istio-system istio-
ingressgateway
kubectl rollout restart deployment --namespace istio-system istio-
egressgateway
```

6. Check the Istio pods are running.

```
kubectl get pods -o wide -A
```

7. Validate the Istio installation.

```
kubectl get pods -A -o yaml | grep image: | grep proxyv2 | grep -i cont
```

The output contains the `image` option for the Istio pods, which shows the Istio version, is now `proxyv2:1.20.5`. For example:

```
image: container-registry.oracle.com/olcne/proxyv2:1.20.5
image: container-registry.oracle.com/olcne/proxyv2:1.20.5
image: container-registry.oracle.com/olcne/proxyv2:1.20.5
image: container-registry.oracle.com/olcne/proxyv2:1.20.5
```

9
Upgrade Kubernetes

Upgrade the Kubernetes cluster to use the latest available Kubernetes release.

Upgrade the Kubernetes cluster to the next available minor Kubernetes version. For example, if the cluster is running Kubernetes Release 1.29, upgrade to Kubernetes Release 1.30. Then repeat the upgrade steps to upgrade to the next minor Kubernetes release, 1.31, and repeat until you're running the latest available version. For information on the latest available Kubernetes release, see Oracle Cloud Native Environment: Release Notes. For more information on upgrading Kubernetes minor releases, see Oracle Cloud Native Environment: Kubernetes Clusters.

1. Set the target Kubernetes version.

   Use `ocne cluster stage` command to stage the target Kubernetes version. The syntax to use is:

   ```
   ocne cluster stage
   [{-c|--config} path]
   [{-r|--os-registry} registry]
   [{-t|--transport} transport]
   {-v|--version} version
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne cluster stage --version 1.30
   ```

2. Confirm an update is available for the cluster nodes.

   Use the `ocne cluster info` command to confirm the nodes are staged with an updated OCK image. Use the syntax:

   ```
   ocne cluster info
   [{-N|--nodes}] nodename, ...
   [{-s|--skip-nodes }]
   ```

   For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

   For example:

   ```
   ocne cluster info
   ```

   When an OCK image is available, staged, and ready to install, the output of this command shows the `Update Available` field to be `true` for a node.

3. Update the control plane nodes.

   Update the control plane nodes, one node at a time, with the staged OCK image.

Use the `ocne node update` command to update each node. Use the syntax:

```
ocne node update
[{-d|--delete-emptydir-data}]
[{-c|--disable-eviction}]
{-N|--node} name
[{-p|--pre-update-mode} mode]
[{-t|--timeout} minutes]
```

For more information on the syntax options, see Oracle Cloud Native Environment: CLI.

For example:

```
ocne node update --node mynode
```

Replace `mynode` with the name of the control plane node.

> **Tip**
>
> After each node is updated, use the `ocne cluster info` command to check the update is complete. Node updates are asynchronous. The update is complete only when the output of this command reports an update is no longer available for a node.

4. Update the worker nodes.

   Use the `ocne node update` command to update each worker node.

5. Confirm Kubernetes has been upgraded.

   Use the `kubectl get nodes` command to confirm all nodes have been upgraded and are listed with the updated Kubernetes version.

   ```
   kubectl get nodes
   ```