Oracle Cloud Native Environment

Installation for Release 1.8

F87562-03
March 2024

Oracle Cloud Native Environment Installation for Release 1.8,

F87562-03

Contents

3 Installing Oracle Cloud Native Environment

4 Creating an Environment

5 Installing Modules

6 Configuring Services

Preface

This document contains information about Oracle Cloud Native Environment. It includes information on installing and configuring Oracle Cloud Native Environment.

Documentation License

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

Host Requirements

This chapter describes the hardware and OS requirements for the hosts in Oracle Cloud Native Environment.

Hardware Requirements

Oracle Cloud Native Environment is a clustered environment that requires more than one node to form a cluster. You can install Oracle Cloud Native Environment on any of the following server types:

- Bare-metal server
- Oracle Cloud Infrastructure bare-metal instance
- Oracle Cloud Infrastructure virtual instance
- Oracle Linux Kernel-based Virtual Machine (KVM) instance
- Oracle Private Cloud Appliance virtual instance
- Oracle Private Cloud at Customer virtual instance

Oracle Cloud Native Environment is available on hardware that uses 64-bit x86 or 64-bit ARM processors. Oracle Cloud Native Environment automatically detects the processor type during an installation and no extra installation steps are required. You can have mixed processor types in a Kubernetes cluster, so you can use 64-bit x86 and 64-bit ARM nodes in the same cluster.

> **Important:**
>
> KubeVirt and Kata Containers require bare metal ARM hosts. Nested virtualization is needed on virtual hosts to create virtual machines using the KubeVirt module, and to create Kata Containers. Nested virtualization isn't available on virtual ARM instances.

Oracle Cloud Native Environment doesn't require specific hardware, but certain operations are CPU and memory intensive. For a list of certified bare-metal servers, see the Oracle Linux Hardware Certification List at:

https://linux.oracle.com/hardware-certifications

For information on the current Oracle x86 Servers, see:

https://www.oracle.com/servers/x86/

For information on 64-bit ARM compute nodes on Oracle Cloud Infrastructure, see:

https://www.oracle.com/cloud/compute/arm/

For information on creating an Oracle Linux KVM instance, see Oracle® Linux: KVM User's Guide.

The installation instructions for Oracle Private Cloud Appliance and Oracle Private Cloud at Customer, and information about the Oracle Cloud Native Environment releases that can be installed, are available in the Oracle Private Cloud Appliance and Oracle Private Cloud at Customer documentation at:

https://docs.oracle.com/en/engineered-systems/private-cloud-appliance/

The hardware requirements listed here are for the absolute minimum to run Oracle Cloud Native Environment. A deployment is highly likely to require nodes with a larger footprint.

For a list of the cloud platforms and hypervisors you can use with Oracle Cloud Native Environment, see Doc ID 2899157.1 in My Oracle Support.

Kubernetes Control Plane Node Hardware

The Kubernetes control plane is the container orchestration layer that exposes the Kubernetes API and interfaces to create and manage the lifecycle of containers. The nodes that form the Kubernetes control plane are referred to as *control plane* nodes. A control plane node is a host that runs the daemons and services needed to manage the cluster and orchestrate containers, such as the Oracle Cloud Native Environment Platform Agent, etcd, the Kubernetes API Server, Scheduler, Controller Manager, and Cloud Controller Manager.

A minimum Kubernetes control plane node configuration is:

- 4 CPU cores (Intel VT-capable CPU)
- 16GB RAM
- 1GB Ethernet NIC
- XFS file system (the default file system for Oracle Linux)
- 40GB hard disk space in the `/var` directory

Kubernetes Worker Node Hardware

A Kubernetes worker node is a host that runs the daemons and services needed to run pods, such as the Platform Agent, kubelet, kube-proxy, CRI-O, RunC, and Kata Runtime.

A minimum Kubernetes worker node configuration is:

- 1 CPU cores (Intel VT-capable CPU)
- 8GB RAM
- 1GB Ethernet NIC
- XFS file system (the default file system for Oracle Linux)
- 15GB hard disk space in the `/var` directory
- XFS mount-point `/var/lib/containers` with dedicated space based on the number of container images going to be saved and leveraged.

Operator Node Hardware

An operator node is a host that contains the Oracle Cloud Native Environment Platform CLI. This node might also include the Oracle Cloud Native Environment Platform API Server.

A minimum operator node configuration is:

- 1 CPU cores (Intel VT-capable CPU)
- 8GB RAM
- 1GB Ethernet NIC
- 15GB hard disk space in the `/var` directory

Kubernetes High Availability Requirements

A minimum high availability (HA) configuration for a Kubernetes cluster is:

- 3 Kubernetes control plane nodes. At least 5 control plane nodes is recommended.
- 2 Kubernetes worker nodes. At least 3 worker nodes is recommended.

> **Important:**
>
> The number of control plane nodes must be an odd number equal to or greater than three, for example, 3, 5, or 7.

Istio Module Requirements

A minimum configuration for deploying the Istio module for Oracle Cloud Native Environment is:

- 1 Kubernetes control plane node
- 2 Kubernetes worker nodes

These requirements are the minimum needed to successfully deploy Istio into a Kubernetes cluster. However, as a cluster expands and more nodes are added, Istio requires more hardware resources.

Rook Module Requirements

The Rook module deploys Ceph as containers to the Kubernetes worker nodes. You need at least three worker nodes in the Kubernetes cluster.

In addition, at least one of these local storage options must be available on the Kubernetes worker nodes:

- Raw devices (no partitions or formatted file systems).
- Raw partitions (no formatted file system).
- LVM Logical Volumes (no formatted file system).
- Persistent Volumes available from a storage class in `block` mode.

> **Tip:**
>
> Use the `lsblk -f` command to ensure no file system is on the device or partition. If the `FSTYPE` field is empty, no file system is on the disk and it can be used with Ceph.

OS Requirements

Oracle Cloud Native Environment is available for the following x86_64 OSs.

Table 1-1 OSs (x86_64)

OS	Release Number	Update Number	Kernel
Oracle Linux	9	Latest and latest-1	Unbreakable Enterprise Kernel Release 7 (UEK R7)
Oracle Linux	9	Latest and latest-1	Red Hat Compatible Kernel (RHCK)
Red Hat Enterprise Linux	9	Latest and latest-1	Red Hat Kernel
Oracle Linux	8	Latest and latest-1	Unbreakable Enterprise Kernel Release 7 (UEK R7)
Oracle Linux	8	Latest and latest-1	Unbreakable Enterprise Kernel Release 6 (UEK R6)
Oracle Linux	8	Latest and latest-1	Red Hat Compatible Kernel (RHCK)
Red Hat Enterprise Linux	8	Latest and latest-1	Red Hat Kernel

2
Prerequisites

This chapter describes the prerequisites for the systems to be used in an installation of Oracle Cloud Native Environment. This chapter also discusses how to enable the repositories to install the Oracle Cloud Native Environment packages.

Enabling Access to the Software Packages

This section contains information on setting up the locations for the OS on which you want to install the Oracle Cloud Native Environment software packages.

Oracle Linux 9

The Oracle Cloud Native Environment packages for Oracle Linux 9 are available on the Oracle Linux yum server in the `ol9_olcne18` repository, or on the Unbreakable Linux Network (ULN) in the `ol9_x86_64_olcne18` channel. However, dependencies exist across other repositories and channels, and these must also be enabled on each system where Oracle Cloud Native Environment is installed.

> **NOT_SUPPORTED:**
>
> Ensure the `ol9_developer` and `ol9_developer_EPEL` yum repositories or ULN channels aren't enabled, and any software from these repositories or channels isn't installed on the systems where Kubernetes runs. Even if you follow the instructions in this document, you might render the platform unstable if these repositories or channels are enabled or software from these channels or repositories is installed on the systems.

Enabling Repositories with the Oracle Linux Yum Server

If you're using the Oracle Linux yum server for system updates, enable the required yum repositories.

To enable the yum repositories:

1. Install the `oracle-olcne-release-el9` release package to install the Oracle Cloud Native Environment yum repository configuration.

   ```
   sudo dnf install oracle-olcne-release-el9
   ```

2. Set up the repositories for the release you want to install.

   Enable the following yum repositories:

   - `ol9_olcne18`
   - `ol9_addons`
   - `ol9_baseos_latest`

- `ol9_appstream`
- `ol9_UEKR7` (if hosts are running UEK R7)

Use the `dnf config-manager` tool to enable the yum repositories. For hosts running UEK R7:

```
sudo dnf config-manager --enable ol9_olcne18 ol9_addons ol9_baseos_latest
ol9_appstream ol9_UEKR7
```

For hosts running RHCK:

```
sudo dnf config-manager --enable ol9_olcne18 ol9_addons ol9_baseos_latest
ol9_appstream
```

3. Disable the yum repositories for previous Oracle Cloud Native Environment releases.

```
sudo dnf config-manager --disable ol9_olcne17
```

4. Disable any developer yum repositories. To list the developer repositories that need to be disabled, use the `dnf repolist` command:

```
sudo dnf repolist --enabled | grep developer
```

Disable the repositories returned using the `dnf config-manager` tool. For example:

```
sudo dnf config-manager --disable ol9_developer
```

Enabling Channels with ULN

If you're registered to use ULN, use the ULN web interface to subscribe the system to the appropriate channels.

To subscribe to the ULN channels:

1. Sign in to https://linux.oracle.com.
2. On the Systems tab, click the link named for the system in the list of registered machines.
3. On the System Details page, click **Manage Subscriptions**.
4. On the System Summary page, select each required channel from the list of available channels and click the arrow to move the channel to the list of subscribed channels.

   Subscribe the system to the following channels:

   - `ol9_x86_64_olcne18`
   - `ol9_x86_64_addons`
   - `ol9_x86_64_baseos_latest`
   - `ol9_x86_64_appstream`
   - `ol9_x86_64_UEKR7` (if hosts are running UEK R7)

   Ensure the systems aren't subscribed to the `ol9_x86_64_olcne17` or `ol9_x86_64_developer` channels.

Oracle Linux 8

The Oracle Cloud Native Environment packages for Oracle Linux 8 are available on the Oracle Linux yum server in the `ol8_olcne18` repository, or on the Unbreakable Linux Network (ULN) in the `ol8_x86_64_olcne18` channel. However, dependencies exist across other repositories and channels, and these must also be enabled on each system where Oracle Cloud Native Environment is installed.

> **NOT_SUPPORTED:**
>
> Ensure the `ol8_developer` or `ol8_developer_EPEL` yum repositories or ULN channels aren't enabled, and any software from these repositories or channels isn't installed on the systems where Kubernetes runs. Even if you follow the instructions in this document, you might render the platform unstable if these repositories or channels are enabled or software from these channels or repositories is installed on the systems.

Enabling Repositories with the Oracle Linux Yum Server

If you're using the Oracle Linux yum server for system updates, enable the required yum repositories.

To enable the yum repositories:

1. Install the `oracle-olcne-release-el8` release package to install the Oracle Cloud Native Environment yum repository configuration.

   ```
   sudo dnf install oracle-olcne-release-el8
   ```

2. Set up the repositories for the release you want to install.

   Enable the following yum repositories:

   - `ol8_olcne18`
   - `ol8_addons`
   - `ol8_baseos_latest`
   - `ol8_appstream`
   - `ol8_kvm_appstream`
   - `ol8_UEKR7` (if hosts are running UEK R7)
   - `ol8_UEKR6` (if hosts are running UEK R6)

   Use the `dnf config-manager` tool to enable the yum repositories. For hosts running UEK R7:

   ```
   sudo dnf config-manager --enable ol8_olcne18 ol8_addons ol8_baseos_latest
   ol8_appstream ol8_kvm_appstream ol8_UEKR7
   ```

   For hosts running UEK R6:

   ```
   sudo dnf config-manager --enable ol8_olcne18 ol8_addons ol8_baseos_latest
   ol8_appstream ol8_kvm_appstream ol8_UEKR6
   ```

For hosts running RHCK:

```
sudo dnf config-manager --enable ol8_olcne18 ol8_addons ol8_baseos_latest
ol8_appstream ol8_kvm_appstream
```

3. Disable the yum repositories for previous Oracle Cloud Native Environment releases.

```
sudo dnf config-manager --disable ol8_olcne17 ol8_olcne16 ol8_olcne15
ol8_olcne14 ol8_olcne13 ol8_olcne12
```

4. Disable any developer yum repositories. To list the developer repositories that need to be disabled, use the `dnf repolist` command:

```
sudo dnf repolist --enabled | grep developer
```

Disable the repositories returned using the `dnf config-manager` tool. For example:

```
sudo dnf config-manager --disable ol8_developer
```

Enabling Channels with ULN

If you're registered to use ULN, use the ULN web interface to subscribe the system to the appropriate channels.

To subscribe to the ULN channels:

1. Sign in to https://linux.oracle.com.
2. On the Systems tab, click the link named for the system in the list of registered machines.
3. On the System Details page, click **Manage Subscriptions**.
4. On the System Summary page, select each required channel from the list of available channels and click the arrow to move the channel to the list of subscribed channels.

   Subscribe the system to the following channels:

   - `ol8_x86_64_olcne18`
   - `ol8_x86_64_addons`
   - `ol8_x86_64_baseos_latest`
   - `ol8_x86_64_appstream`
   - `ol8_x86_64_kvm_appstream`
   - `ol8_x86_64_UEKR7` (if hosts are running UEK R7)
   - `ol8_x86_64_UEKR6` (if hosts are running UEK R6)

   Ensure the systems aren't subscribed to the following channels:

   - `ol8_x86_64_developer`
   - `ol8_x86_64_olcne17`
   - `ol8_x86_64_olcne16`
   - `ol8_x86_64_olcne15`
   - `ol8_x86_64_olcne14`

- `ol8_x86_64_olcne13`
- `ol8_x86_64_olcne12`

Accessing the Oracle Container Registry

The container images that are deployed by the Platform CLI are hosted on the Oracle Container Registry. For more information about the Oracle Container Registry, see the Oracle® Linux: Oracle Container Runtime for Docker User's Guide.

For a deployment to use the Oracle Container Registry, each node within the environment must be provisioned with direct access to the Internet.

You can optionally use an Oracle Container Registry mirror, or create a private registry mirror within the network.

When you create a Kubernetes module you must specify the registry from which to pull the container images. This is set using the `--container-registry` option of the `olcnectl module create` command. If you use the Oracle Container Registry the container registry must be set to:

```
container-registry.oracle.com/olcne
```

If you use a private registry that mirrors the Oracle Cloud Native Environment container images on the Oracle Container Registry, ensure you set the container registry to the domain name and port of the private registry, for example:

```
myregistry.example.com:5000/olcne
```

When you set the container registry to use during an installation, it becomes the default registry from which to pull images during updates and upgrades of the Kubernetes module. You can set a new default value during an update or upgrade using the `--container-registry` option.

Using an Oracle Container Registry Mirror

The Oracle Container Registry has many mirror servers around the world. You can use a registry mirror in a global region to improve download performance of container images. While the Oracle Container Registry mirrors are hosted on Oracle Cloud Infrastructure, they're also accessible external to Oracle Cloud Infrastructure. Using a mirror that's closest to a geographical location results in faster download speeds.

To use an Oracle Container Registry mirror to pull images, use the format:

```
container-registry-region-key.oracle.com/olcne
```

For example, to use the Oracle Container Registry mirror in the US East (Ashburn) region, which has a region key of `IAD`, the registry is set (using the using the `--container-registry` option) to:

```
container-registry-iad.oracle.com/olcne
```

For more information on Oracle Container Registry mirrors and finding the region key for a mirror in a location, see the Oracle Cloud Infrastructure documentation.

Using a Private Registry

Sometimes, nodes within an environment might not be provisioned with direct access to the Internet. In these cases, you can use a private registry that mirrors the Oracle Cloud Native Environment container images on the Oracle Container Registry. Each node requires direct access to the mirror registry host in this scenario.

You can use an existing container registry in a network, or create a private registry using Podman on an Oracle Linux 8 host. If you use an existing private container registry, skip the first step in the following procedure that creates a registry.

To create a private registry:

1. Select an Oracle Linux 8 host to use for the Oracle Container Registry mirror service. The mirror host must have access to the Internet and able to pull images directly from the Oracle Container Registry, or alternately, have access to the correct image files stored locally. Ideally, the host isn't a node within an Oracle Cloud Native Environment, but is accessible to all nodes that are part of the environment.

   On the mirror host, install Podman, and set up a private registry, following the instructions in the Setting up a Local Container Registry section in Oracle® Linux: Podman User's Guide.

2. On the mirror host, enable access to the Oracle Cloud Native Environment software packages. For information on enabling access to the packages, see Enabling Access to the Software Packages.

3. Install the `olcne-utils` package so you have access to the registry mirroring utility.

   ```
   sudo dnf install olcne-utils
   ```

   If you're using an existing container registry in the network that's running on Oracle Linux 7, use `yum` instead of `dnf` to install `olcne-utils`.

4. Copy the required container images from the Oracle Container Registry to the private registry using the `registry-image-helper.sh` script with the required options:

   ```
   registry-image-helper.sh --to host.example.com:5000/olcne
   ```

   Where *host.example.com:5000* is the resolvable domain name and port on which the private registry is available.

   You can optionally use the `--from` option to specify a different registry from which to pull the images. For example, to pull the images from an Oracle Container Registry mirror:

   ```
   registry-image-helper.sh \
   --from container-registry-iad.oracle.com/olcne \
   --to host.example.com:5000/olcne
   ```

   If the host where you're running the script doesn't have access to the Internet, you can replace the `--from` option with the `--local` option to load the container images directly from a local directory. The local directory which contains the images can be either:

   - `/usr/local/share/kubeadm/`

- `/usr/local/share/olcne/`

The image files must be archives in TAR format. All TAR files in the directory are loaded into the private registry when the script is run with the `--local` option.

You can use the `--version` option to specify the Kubernetes version you want to mirror. If not specified, the latest release is used. The available versions you can pull are listed in Release Notes.

Setting up the OS

The following sections describe the requirements that must be met to install and configure Oracle Cloud Native Environment on Oracle Linux systems.

Setting up a Network Time Service

As a clustering environment, Oracle Cloud Native Environment requires that the system time is synchronized across each Kubernetes control plane and worker node within the cluster. Typically, this can be achieved by installing and configuring a Network Time Protocol (NTP) daemon on each node. We recommend installing and setting up the chronyd daemon for this purpose.

The `chronyd` service is enabled and started by default on Oracle Linux systems.

Systems running on Oracle Cloud Infrastructure are configured to use the `chronyd` time service by default, so you don't need to add or configure NTP if you're installing into an Oracle Cloud Infrastructure environment.

Disabling Swap

You must disable swap on the Kubernetes control plane and worker nodes. To disable swap, enter:

```
sudo swapoff -a
```

To make this permanent over reboots, edit the `/etc/fstab` file to remove or comment out any swap disks. For example, you can consider using commands similar to those shown in the following steps:

1. Check contents of the `/etc/fstab` file before any change:

   ```
   sudo cat /etc/fstab
   ```

2. Make a backup of `/etc/fstab`.

   ```
   sudo cp /etc/fstab /etc/fstab_copy
   ```

3. Comment out swap disks from the `/etc/fstab` file:

   ```
   sudo sed -i '/\bswap\b/s/^/#/' /etc/fstab
   ```

4. Check contents of `/etc/fstab` after the change:

   ```
   sudo cat /etc/fstab
   ```

Setting up the Network

This section contains information about the networking requirements for Oracle Cloud Native Environment nodes.

The following table shows the network ports used by the services in a deployment of Kubernetes in an environment.

From Node Type	To Node Type	Port	Protocol	Reason
Worker	Operator	8091	TCP(6)	Platform API Server
Control plane	Operator	8091	TCP(6)	Platform API Server
Control plane	Control plane	2379-2380	TCP(6)	Kubernetes etcd (highly available clusters)
Operator	Control plane	6443	TCP(6)	Kubernetes API server
Worker	Control plane	6443	TCP(6)	Kubernetes API server
Control plane	Control plane	6443	TCP(6)	Kubernetes API server
Control plane	Control plane	6444	TCP(6)	Alternate Kubernetes API server (highly available clusters)
Operator	Control plane	8090	TCP(6)	Platform Agent
Control plane	Control plane	10250 10251 10252 10255	TCP(6) TCP(6) TCP(6) TCP(6)	Kubernetes `kubelet` API server Kubernetes `kube-scheduler` (highly available clusters) Kubernetes `kube-controller-manager` (highly available clusters) Kubernetes `kubelet` API server for read-only access with no authentication

From Node Type	To Node Type	Port	Protocol	Reason
Control plane	Control plane	8472	UDP(11)	Flannel
Control plane	Worker	8472	UDP(11)	Flannel
Worker	Control plane	8472	UDP(11)	Flannel
Worker	Worker	8472	UDP(11)	Flannel
Control plane	Control plane	N/A	VRRP(112)	Keepalived for Kubernetes API server (highly available clusters)
Operator	Worker	8090	TCP(6)	Platform Agent
Control plane	Worker	10250 10255	TCP(6) TCP(6)	Kubernetes `kubelet` API server Kubernetes `kubelet` API server for read-only access with no authentication

The following sections show you how to set up the network on each node to enable the communication between nodes in an environment.

> **Important:**
>
> In addition to opening network ports on the operator and Kubernetes nodes, if you're using an external firewall (hardware or software based), ensure the ports in the previous table are open on the external firewall before you perform an installation.

Setting up the Firewall Rules

Oracle Linux installs and enables `firewalld`, by default. You can install Oracle Cloud Native Environment with `firewalld` enabled, or you can disable it and use another firewall solution. This sections shows you how to set up the firewall rules to enable `firewalld`.

> **Important:**
>
> Calico requires the `firewalld` service to be disabled. If you're installing Calico as the Kubernetes CNI for pods, you don't need to configure the networking ports as shown in this section. See Calico Module for information on disabling `firewalld` and how to install Calico.

If you want install with `firewalld` enabled, the Platform CLI notifies you of any rules that you need to add during the deployment of the Kubernetes module. The Platform CLI also provides the commands to run to change the firewall configuration to meet the requirements.

Ensure that all required ports are open. The ports required for a Kubernetes deployment are:

- 2379/tcp: Kubernetes `etcd` server client API (on control plane nodes in highly available clusters)
- 2380/tcp: Kubernetes `etcd` server client API (on control plane nodes in highly available clusters)
- 6443/tcp: Kubernetes API server (control plane nodes)
- 8090/tcp: Platform Agent (control plane and worker nodes)
- 8091/tcp: Platform API Server (operator node)
- 8472/udp: Flannel overlay network, VxLAN backend (control plane and worker nodes)
- 10250/tcp: Kubernetes `kubelet` API server (control plane and worker nodes)
- 10251/tcp: Kubernetes `kube-scheduler` (on control plane nodes in highly available clusters)
- 10252/tcp: Kubernetes `kube-controller-manager` (on control plane nodes in highly available clusters)
- 10255/tcp: Kubernetes `kubelet` API server for read-only access with no authentication (control plane and worker nodes)

The commands to open the ports and to set up the firewall rules are provided.

Non-HA Cluster Firewall Rules

For a cluster with a single control plane node, the following ports are required to be open in the firewall.

Operator Node

On the operator node, run:

```
sudo firewall-cmd --add-port=8091/tcp --permanent
```

Restart the firewall for these rules to take effect:

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

Worker Nodes

On the Kubernetes worker nodes run:

```
sudo firewall-cmd --zone=trusted --add-interface=cni0 --permanent
sudo firewall-cmd --add-port=8090/tcp --permanent
sudo firewall-cmd --add-port=10250/tcp --permanent
sudo firewall-cmd --add-port=10255/tcp --permanent
sudo firewall-cmd --add-port=8472/udp --permanent
```

Restart the firewall for these rules to take effect:

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

Control Plane Nodes

On the Kubernetes control plane nodes run:

```
sudo firewall-cmd --zone=trusted --add-interface=cni0 --permanent
sudo firewall-cmd --add-port=8090/tcp --permanent
sudo firewall-cmd --add-port=10250/tcp --permanent
sudo firewall-cmd --add-port=10255/tcp --permanent
sudo firewall-cmd --add-port=8472/udp --permanent
sudo firewall-cmd --add-port=6443/tcp --permanent
```

Restart the firewall for these rules to take effect:

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

Highly Available Cluster Firewall Rules

For a highly available cluster, open all the firewall ports as described in Non-HA Cluster Firewall Rules, along with the following **extra** ports on the control plane nodes.

On the Kubernetes control plane nodes run:

```
sudo firewall-cmd --add-port=10251/tcp --permanent
sudo firewall-cmd --add-port=10252/tcp --permanent
sudo firewall-cmd --add-port=2379/tcp --permanent
sudo firewall-cmd --add-port=2380/tcp --permanent
```

Restart the firewall for these rules to take effect:

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

Setting up Other Network Options

This section contains information on other network related configuration that affects an Oracle Cloud Native Environment deployment. You might not need to make changes from this section, but they're provided to help you understand any issues you might meet related to network configuration.

Internet Access

The Platform CLI checks it can access the container registry, and possibly other Internet resources, to pull any required container images. Unless you intend to set up a local registry mirror for container images, the systems where you intend to install Oracle Cloud Native Environment must either have direct internet access, or must be configured to use a proxy.

nftables Rule on Oracle Linux 9

If you're using Oracle Linux 9 hosts and you have firewalld enabled, you must add an nftables rule on the Kubernetes nodes.

If you don't need to persist the rule over reboots, set up the rule on each Kubernetes node using:

```
sudo nft add rule inet firewalld filter_FORWARD_POLICIES_post accept
```

If you do need to persist the rule over host reboots (recommended), set up a rule file and create a systemd service that's enabled. On each Kubernetes node, run the following.

1. Create a file named `/etc/nftables/forward-policies.nft` which contains the rule by entering:

```
sudo sh -c "cat > /etc/nftables/forward-policies.nft << EOF
flush chain inet firewalld filter_FORWARD_POLICIES_post

table inet firewalld {
        chain filter_FORWARD_POLICIES_post {
                accept
        }
}
EOF"
```

2. Create a file named `/etc/systemd/system/forward-policies.service` which contains the systemd service by entering:

```
sudo sh -c "cat > /etc/systemd/system/forward-policies.service <<
EOF
[Unit]
Description=Idempotent nftables rules for forward-policies
PartOf=firewalld.service

[Service]
ExecStart=/sbin/nft -f /etc/nftables/forward-policies.nft
ExecReload=/sbin/nft -f /etc/nftables/forward-policies.nft
Restart=always
StartLimitInterval=0
RestartSec=10

[Install]
WantedBy=multi-user.target
EOF"
```

3. Enable and restart the rule service:

```
sudo systemctl enable forward-policies.service
sudo systemctl restart forward-policies.service
```

Flannel Network

The Platform CLI configures a Flannel network as the network fabric used for communications between Kubernetes pods. This overlay network uses VxLANs for network connectivity. For more information on Flannel, see the upstream documentation.

By default, the Platform CLI creates a network in the `10.244.0.0/16` range to host this network. The Platform CLI provides an option to set the network range to another range, if required, during installation. Systems in an Oracle Cloud Native Environment deployment must not have any network devices configured for this reserved IP range.

br_netfilter Module

The Platform CLI checks whether the `br_netfilter` module is loaded and exits if it's not available. This module is required to enable transparent masquerading and for Virtual Extensible LAN (VxLAN) traffic for communication between Kubernetes pods across the cluster. If you need to check whether it's loaded, run:

```
sudo lsmod|grep br_netfilter
```

If you see the output similar to the following, the `br_netfilter` module is loaded.

```
br_netfilter           24576  0
bridge                155648  2 br_netfilter,ebtable_broute
```

Kernel modules are loaded as they're needed, and it's unlikely that you need to load this module manually. You can load the module manually and add it as a permanent module by running:

```
sudo modprobe br_netfilter
sudo sh -c 'echo "br_netfilter" > /etc/modules-load.d/br_netfilter.conf'
```

Bridge Tunable Parameters

Kubernetes requires that packets traversing a network bridge are processed for filtering and for port forwarding. To achieve this, tunable parameters in the kernel bridge module are automatically set when the `kubeadm` package is installed and a sysctl file is created at `/etc/sysctl.d/k8s.conf` that contains the following lines:

```
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
```

If you change this file, or create anything similar yourself, run the following command to load the bridge tunable parameters:

```
sudo /sbin/sysctl -p /etc/sysctl.d/k8s.conf
```

Network Address Translation

Network Address Translation (NAT) is sometimes required when one or more Kubernetes worker nodes in a cluster are behind a NAT gateway. For example, you might want to have a control plane node in a secure company network while having other worker nodes in a publicly accessible demilitarized zone which is less secure. The control plane node would access the worker nodes through the worker node's NAT gateway. Or you might have a worker node in a legacy network that you want to use in the cluster that's primarily on a newer network. The NAT gateway, in these cases, translates requests for an IP address accessible to the Kubernetes cluster into the IP address on the subnet behind the NAT gateway.

> **Note:**
>
> Only worker nodes can be behind a NAT. Control plane nodes can't be behind a NAT.

Regardless of what switches or network equipment you use to set up the NAT gateway, you must configure the following for a node behind a NAT gateway:

- The node's interface behind the NAT gateway must have an public IP address using the `/32` subnet mask that's reachable by the Kubernetes cluster. The `/32` subnet restricts the subnet to one IP address, so that all traffic from the Kubernetes cluster flows through this public IP address.
- The node's interface must also include a private IP address behind the NAT gateway that the switch uses NAT tables to match the public IP address to.

For example, you can use the following command to add the reachable IP address on the `ens5` interface:

```
sudo ip addr add 192.168.64.6/32 dev ens5
```

You can then use the following command to add the private IP address on the same interface:

```
sudo ip addr add 192.168.192.2/18 dev ens5
```

Setting FIPS Mode

You can optionally configure Oracle Cloud Native Environment operator, control plane, and worker hosts to run in Federal Information Processing Standards (FIPS) mode as described in Oracle Linux 9: Installing and Configuring FIPS Mode or Oracle Linux 8: Enhancing System Security.

Oracle Cloud Native Environment uses the cryptographic binaries of OpenSSL from Oracle Linux when the host runs in FIPS mode.

Setting Up SSH Key-based Authentication

Set up SSH key-based authentication for the user that's to be used to run the Platform CLI (`olcnectl`) installation commands to enable login from the operator node to each Kubernetes node and to the Platform API Server node.

The following steps show one method of setting up SSH key-based authentication.

1. Generate the private and public key pair. On the operator node, run `ssh-keygen` as the user that you use to run `olcnectl` commands. Don't create a passphrase for the key (press `<Enter>` when prompted for a passphrase). For example:

   ```
   ssh-keygen
   ```

Output similar to the following is displayed:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):<Enter>
Enter passphrase (empty for no passphrase): <Enter>
Enter same passphrase again: <Enter>
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
...
```

2. Verify the location of the private and public key pair. Verify the private and public key pair have been created at the location reported in the `ssh-keygen` command output:

   ```
   ls -l /home/user/.ssh/
   ```

   Output similar to the following is displayed:

   ```
   ...
   -rw-------. 1 user user 2643 Jan 10 14:55 id_rsa
   -rw-r--r--. 1 user user  600 Jan 10 14:55 id_rsa.pub
   ...
   ```

   The public key is indicated by the file with the "`.pub`" extension.

3. Set up the public key on the target nodes. Add the contents of the public key to the `$HOME/.ssh/authorized_keys` file on each target node for the user for which the key-based SSH is being set up.

   On the operator node, run the `ssh-copy-id` command. The syntax is:

   ```
   ssh-copy-id user@host
   ```

   When prompted you enter the user's password for the host. After the command successfully completes, the public key's contents have been added to the copy of the user's `$HOME/.ssh/authorized_keys` file on the remote host.

   The following example shows how command `ssh-copy-id` can be used to add the public key to the `authorized_keys` file for user on host `192.0.2.255` :

   ```
   ssh-copy-id user@192.0.2.255
   ```

4. Verify the user has SSH key-based access from the operator node. On the operator node, use `ssh` to connect to each of the other nodes and confirm login succeeds without being prompted for a password.

   For example, confirm key-based SSH access by running the `ssh` command on the operator node as follows:

   ```
   ssh user@192.0.2.255
   ```

For more information on setting up SSH key-based authentication, see Oracle Linux: Connecting to Remote Systems With OpenSSH.

3

Installing Oracle Cloud Native Environment

This chapter discusses how to prepare the nodes to be used in an Oracle Cloud Native Environment deployment. When the nodes are prepared, they must be installed with the Oracle Cloud Native Environment software packages. When the nodes are set up with the software, you can use the Platform CLI to perform a deployment of a Kubernetes cluster and optionally install other modules.

This chapter shows you how to perform the steps to set up the hosts and install the Oracle Cloud Native Environment software, ready to perform a deployment of modules. When you have set up the nodes, deploy the Kubernetes module to install a Kubernetes cluster using the steps in Kubernetes Module.

Installation Overview

The high level overview of setting up Oracle Cloud Native Environment is described in this section.

To install Oracle Cloud Native Environment:

1. **Prepare the operator node**: An *operator* node is a host to be used to perform and manage the deployment of environments. The operator node must be set up with the Platform API Server, and the Platform CLI (`olcnectl`).
2. **Prepare the Kubernetes nodes**: The Kubernetes control plane and worker nodes must to be set up with the Platform Agent.
3. **Set up a load balancer**: If you're deploying a highly available Kubernetes cluster, set up a load balancer. You can set up an external load balancer, or use the container-based load balancer deployed by the Platform CLI.
4. **Set up X.509 Certificates**: X.509 Certificates are used to provide secure communication between the Kubernetes nodes. You must set up the certificates before you create an environment and perform a deployment.
5. **Start the services**: Start the Platform API Server and Platform Agent services on nodes using the X.509 Certificates.
6. **Create an environment**: Create an environment into which you can install the Kubernetes module and any other optional modules.
7. **Deploy modules**: Deploy the Kubernetes module and any other optional modules.

Setting up the Nodes

This section discusses setting up nodes to use in an Oracle Cloud Native Environment. The nodes are used to form a Kubernetes cluster.

An *operator* node is used to perform the deployment of the Kubernetes cluster using the Platform CLI and the Platform API Server. An operator node might be a node in the Kubernetes cluster, or a separate host. In examples in this book, the operator node is a separate host, and not part of the Kubernetes cluster.

On each Kubernetes node (both control plane and worker nodes) the Platform Agent must be installed. Before you set up the Kubernetes nodes, you must prepare them. For information on preparing the nodes, see Prerequisites.

During the installation of the required packages, an `olcne` user is created. This user is used to start the Platform API Server or Platform Agent services and has the minimum OS privileges to perform that task. Don't use the `olcne` user for any other purpose.

Setting up the Operator Node

This section discusses setting up the operator node. The *operator* node is a host that's used to perform and manage the deployment of environments, including deploying the Kubernetes cluster.

To set up the operator node:

1. On the operator node, install the Platform CLI, Platform API Server, and utilities.

   ```
   sudo dnf install olcnectl olcne-api-server olcne-utils
   ```

2. Enable the `olcne-api-server` service, but do *not* start it. The `olcne-api-server` service is started when you configure the X.509 Certificates.

   ```
   sudo systemctl enable olcne-api-server.service
   ```

   For information on configuration options for the Platform API Server, see Configuring the Platform API Server.

Setting up Kubernetes Nodes

This section discusses setting up the nodes to use in a Kubernetes cluster. Perform these steps on both Kubernetes control plane and worker nodes.

To set up the Kubernetes nodes:

1. On each node to be added to the Kubernetes cluster, install the Platform Agent package and utilities.

   ```
   sudo dnf install olcne-agent olcne-utils
   ```

2. Enable the `olcne-agent` service, but do *not* start it. The `olcne-agent` service is started when you configure the X.509 Certificates.

   ```
   sudo systemctl enable olcne-agent.service
   ```

   For information on configuration options for the Platform Agent, see Configuring the Platform Agent.

3. If you use a proxy server, configure it with CRI-O. On each Kubernetes node, create a CRI-O `systemd` configuration directory:

   ```
   sudo mkdir /etc/systemd/system/crio.service.d
   ```

Create a file named `proxy.conf` in the directory, and add the proxy server information. For example:

```
[Service]
Environment="HTTP_PROXY=http://proxy.example.com:3128"
Environment="HTTPS_PROXY=https://proxy.example.com:3128"
Environment="NO_PROXY=mydomain.example.com"
```

If you're also installing Calico (as a module or as the Kubernetes Container Network Interface), or the Multus module, add the Kubernetes service IP (the default is `10.96.0.1`) to the `NO_PROXY` variable:

```
Environment="NO_PROXY=mydomain.example.com,10.96.0.1"
```

4. If the `docker` service is running, stop, and disable it.

```
sudo systemctl disable --now docker.service
```

5. If the `containerd` service is running, stop, and disable it.

```
sudo systemctl disable --now containerd.service
```

Setting up a Load Balancer for Highly Available Clusters

A highly available (HA) cluster needs a load balancer to provide high availability of control plane nodes. A load balancer communicates with the Kubernetes API Server on the control plane nodes.

The methods of setting up a load balancer to create an HA cluster are:

- Using an external load balancer instance.
- Using a load balancer provided by a cloud infrastructure, for example an Oracle Cloud Infrastructure load balancer.
- Using the internal load balancer that can be deployed by the Platform CLI on the control plane nodes.

Setting up an External Load Balancer

To use an external load balancer implementation, it must be set up and ready to use before you perform an HA cluster deployment. The load balancer hostname and port is entered as an option when you create the Kubernetes module. The load balancer must be set up with the following configuration:

- The listener listening on TCP port 6443.
- The distribution set to round robin.
- The target set to TCP port 6443 on the control plane nodes.
- The health check set to TCP.

For more information on setting up an external load balancer, see Oracle Linux 9: Setting Up Load Balancing or Oracle Linux 8: Setting Up Load Balancing.

Setting up a Load Balancer on Oracle Cloud Infrastructure

To set up a load balancer on Oracle Cloud Infrastructure:

1. Sign-in to Oracle Cloud Infrastructure.
2. Create a load balancer.
3. Add a backend set to the load balancer using weighted round robin. Set the health check to be TCP port 6443.
4. Add the control plane nodes to the backend set. Set the port for the control plane nodes to port 6443.
5. Create a listener for the backend set using TCP port 6443.

For more information on setting up a load balancer in Oracle Cloud Infrastructure, see the Oracle Cloud Infrastructure documentation.

Setting up the Internal Load Balancer

> **Important:**
>
> Using the internal load balancer is **not** recommended for production deployments. Instead, use a correctly configured load-balancer that's outside the Kubernetes cluster, for example an own external load balancer, or a load balancer provided by a cloud infrastructure, such as an Oracle Cloud Infrastructure load balancer.

To use the internal load balancer deployed by the Platform CLI, you need to perform the following steps to prepare the control plane nodes.

To prepare control plane nodes for the load balancer deployed by the Platform CLI:

1. Set up the control plane nodes as described in Setting up Kubernetes Nodes.
2. Use the `--virtual-ip` option when creating the Kubernetes module to nominate a virtual IP address that can be used for the primary control plane node. This IP address must not be in use on any node and is assigned dynamically to the control plane node assigned as the primary controller by the load balancer. If the primary node fails, the load balancer reassigns the virtual IP address to another control plane node, and that, in turn, becomes the primary node.

   > **Tip:**
   >
   > If you're deploying to Oracle Cloud Infrastructure virtual instances, you can assign a secondary private IP address to the VNIC on a control plane node to create a virtual IP address. Ensure you list this control plane node first when creating the Kubernetes module. For more information on secondary private IP addresses, see the Oracle Cloud Infrastructure documentation.

3. On each control plane node, open port 6444. When you use a virtual IP address, the Kubernetes API server port is changed from the default of 6443 to 6444. The load balancer listens on port 6443 and receives the requests and passes them to the Kubernetes API server.

```
sudo firewall-cmd --add-port=6444/tcp
sudo firewall-cmd --add-port=6444/tcp --permanent
```

4. On each control plane node, enable the Virtual Router Redundancy Protocol (VRRP) protocol:

```
sudo firewall-cmd --add-protocol=vrrp
sudo firewall-cmd --add-protocol=vrrp --permanent
```

Setting up Certificates for Kubernetes Nodes

Communication between the Kubernetes nodes is secured using X.509 certificates.

Before you deploy Kubernetes, you need to configure the X.509 certificates used to manage the communication between the nodes. You can use:

- **Vault**: The certificates are managed using the HashiCorp Vault secrets manager. Certificates are created *during* the deployment of the Kubernetes module. You need to create a token authentication method for Oracle Cloud Native Environment.
- **CA Certificates**: Using certificates signed by a trusted Certificate Authority (CA), and copied to each Kubernetes node *before* the deployment of the Kubernetes module. These certificates are unmanaged and must be renewed and updated manually.
- **Private CA Certificates**: Using generated certificates, signed by a private CA you set up, and copied to each Kubernetes node *before* the deployment of the Kubernetes module. These certificates are unmanaged and must be renewed and updated manually. A script is provided to help you set this up.

A software-based secrets manager is recommended to manage these certificates. The HashiCorp Vault secrets manager can be used to generate, assign, and manage the certificates. We recommend you implement an instance of Vault, setting up the appropriate security for the environment.

For more information on installing and setting up Vault, see the Hashicorp documentation.

If you don't want to use Vault, you can use certificates, signed by a trusted CA, and copied to each node. A script is provided to generate a private CA to generate certificates for each node. This script also gives you the commands needed to copy the certificates to the nodes.

Setting up Vault Authentication

To configure Vault for use with Oracle Cloud Native Environment, set up a Vault token with the following properties:

- A PKI secret engine with a CA certificate or intermediate, at `olcne_pki_intermediary`.
- A role under that PKI, named `olcne`, configured to not require a common name, and allow any name.
- A token authentication method and policy that attaches to the `olcne` role and can request certificates.

For information on setting up the Vault PKI secrets engine to generate dynamic X.509 certificates, see:

https://developer.hashicorp.com/vault/docs/secrets/pki

For information on creating Vault tokens, see:

https://developer.hashicorp.com/vault/docs/commands/token/create

Setting up CA Certificates

This section shows you how to use certificates signed by a trusted CA, without using a secrets manager such as Vault. To use certificates, copy them to all Kubernetes nodes, and to the Platform API Server node.

To ensure the Platform Agent on each Kubernetes node, and the Platform API Server have access to certificates, copy them into the `/etc/olcne/certificates/` directory on each node. This is the default location of the certificates and is used when setting up the Platform Agent and Platform API Server, and when creating an environment. You can use another location for certificates, but this means you need to specify the location for each certificate and key when you're setting up the services, and the environment.

> **Tip:**
>
> You can use the `olcnectl certificates copy` command to copy the certificates to the Kubernetes nodes.

The default location for certificates and keys, and the location used in this book, is the `/etc/olcne/certificates/` directory on each node.

- **CA Certificate**: `/etc/olcne/certificates/ca.cert`
- **Node Key**: `/etc/olcne/certificates/node.key`
- **Node Certificate**: `/etc/olcne/certificates/node.cert`

Setting up Private CA Certificates

This section shows you how to create a private CA, and use that to generate signed certificates for the nodes. This section also contains information on copying the certificates to the nodes. This section also contains information on generating certificates for nodes that you want to scale into a Kubernetes cluster.

Distribute Node Certificates

Use the Platform CLI to generate and distribute signed certificates to the Kubernetes nodes. You can use a CA Certificate or create one automatically. These certificates are used when starting the Platform API Server and Platform agent on nodes to secure communication between the Kubernetes nodes.

Before you create the certificates, ensure the user on the operator node that's creating the certificates is a member of the `olcne` group. Use the syntax:

```
sudo usermod -a -G olcne username
```

For example, to add the `olcne` group to the `oracle` user, on the operator node run:

```
sudo usermod -a -G olcne oracle
```

> **Important:**
>
> When you add a group to a user, you must log out of the terminal session, and back in again. This is required to apply the change.

On the operator node, use the `olcnectl certificates distribute` command to generate and distribute private CA and certificates to the Kubernetes nodes. The syntax to use is:

```
olcnectl certificates distribute
[--byo-ca-cert certificate-path]
[--byo-ca-key key-path]
[--cert-dir certificate-directory]
[--cert-request-common-name common_name]
[--cert-request-country country]
[--cert-request-locality locality]
[--cert-request-organization organization]
[--cert-request-organization-unit organization-unit]
[--cert-request-state state]
[{-h|--help}]
[{-n|--nodes} nodes]
[--one-cert]
[{-R|--remote-command} remote-command]
[{-i|--ssh-identity-file} file_location]
[{-l|--ssh-login-name} username]
[--timeout minutes]
```

Provide the nodes for which you want to create certificates using the `--nodes` option. Create a certificate for each node that runs the Platform API Server or Platform Agent. This means you must create certificates for the operator node, and for each Kubernetes node.

> **Note:**
>
> If you're deploying a highly available Kubernetes cluster using a virtual IP address, you don't need to create a certificate for a virtual IP address.

For example:

```
olcnectl certificates distribute \
--nodes
operator.example.com,control1.example.com,worker1.example.com,worker2.example
.com
```

To set up the information for the private CA, use the `--cert-request-*` options. The following example also includes options to set the SSH login information for nodes using the `--ssh-*` options.

```
olcnectl certificates distribute \
--nodes
operator.example.com,control1.example.com,worker1.example.com,worker2.e
xample.com \
--cert-request-common-name cloud.example.com \
--cert-request-country US \
--cert-request-locality "My Town" \
--cert-request-organization "My Company" \
--cert-request-organization-unit "My Company Unit" \
--cert-request-state "My State" \
--ssh-identity-file ~/.ssh/id_rsa \
--ssh-login-name oracle
```

If you have a CA Certificate and private key to generate certificates, you can provide it using the `--byo-ca-cert` and `--byo-ca-key` options, for example:

```
olcnectl certificates distribute \
--nodes
operator.example.com,control1.example.com,worker1.example.com,worker2.e
xample.com \
--byo-ca-cert $HOME/certificates/ca/ca.cert \
--byo-ca-key $HOME/certificates/ca/ca.key
```

The certificate files for each node are generated and saved in the `$HOME/certificates/` directory, in a directory for each hostname. If you used the `--cert-dir` option, they're saved to the directory you specified.

The CA Certificate and private key are generated to the `$HOME/certificates/ca/` directory. If you used the `--cert-dir` option, they're saved to the directory you specified. If you used a CA Certificate and key using the `--byo-ca-*` options, they're not created, or saved.

The certificates for each node are copied to the `/etc/olcne/certificates/` directory on each Kubernetes node using SSH. This is the default directory for the location of the certificates when you start the Platform API Server and Platform Agent.

Distribute Extra Node Certificates

Use the Platform CLI to generate and distribute signed certificates to the nodes using an existing CA.

This is useful to generate and distribute certificates for any extra nodes that you want to add to a Kubernetes cluster.

On the operator node, use the `olcnectl certificates distribute` command to generate and distribute private CA and certificates to the Kubernetes nodes. The syntax to use is:

```
olcnectl certificates distribute
[--byo-ca-cert certificate-path]
```

```
[--byo-ca-key key-path]
[--cert-dir certificate-directory]
[--cert-request-common-name common_name]
[--cert-request-country country]
[--cert-request-locality locality]
[--cert-request-organization organization]
[--cert-request-organization-unit organization-unit]
[--cert-request-state state]
[{-h|--help}]
[{-n|--nodes} nodes]
[--one-cert]
[{-R|--remote-command} remote-command]
[{-i|--ssh-identity-file} file_location]
[{-l|--ssh-login-name} username]
[--timeout minutes]
```

Provide the nodes for which you want to create certificates using the `--nodes` option.

Provide the location of the existing CA Certificate using the `--byo-ca-cert` option.

You can use the same CA certificate and private key you used to generate the Kubernetes node certificates by using the `--byo-ca-cert` and `--byo-ca-key` options.

For example:

```
olcnectl certificates distribute \
--nodes worker3.example.com,worker4.example.com \
--byo-ca-cert $HOME/certificates/ca/ca.cert \
--byo-ca-key $HOME/certificates/ca/ca.key
```

The location of the CA Certificate and private key might be different if you used the `--cert-dir` option of the `olcnectl certificates distribute` command when creating the original certificates.

The certificate files for each node are generated and saved in the `$HOME/certificates/` directory, in a directory for each hostname.

The certificates for each node are copied to the `/etc/olcne/certificates/` directory on each Kubernetes node using SSH. This is the default directory for the location of the certificates when you start the Platform API Server and Platform Agent.

Setting up Certificates for the Platform CLI to the Platform API Server

Communication between the Platform CLI and Platform API Server is secured using X.509 certificates.

We recommend you configure the X.509 certificates used to manage the communication between the Platform CLI and the Platform API Server. You can use:

- **Vault**: The certificates are managed using the HashiCorp Vault secrets manager.
- **CA Certificates**: Using a CA Certificate, signed by a trusted Certificate Authority.

- **Private CA Certificates**: Using generated certificates, signed by a private CA you set up. These certificates are unmanaged and must be renewed and updated manually.

The certificates need to be on the node that contains the Platform CLI (this is most likely the operator node). The name of the directory reflects the name of the host that contains Platform API Server (again, this is most likely the operator node). The format of the directory for the certificates must be:

```
$HOME/.olcne/certificates/api_server_hostname:port
```

The `api_server_hostname` is the hostname of the node where the Platform API Server is installed, and the `port` is the port to access the Platform API Server (the default is `8091`). This is the default directory for the Platform API Server keys, and where the Platform CLI checks for access to the Platform API Server without any extra configuration. For example:

```
$HOME/.olcne/certificates/operator.example.com:8091
```

Generate Certificates for the Platform CLI to the Platform API Server

Use the Platform CLI to generate signed certificates for the Platform CLI to access the Platform API Server.

On the operator node, use the `olcnectl certificates generate` command to generate certificates for the Platform CLI to access the Platform API Server. The syntax to use is:

```
olcnectl certificates generate
[--byo-ca-cert certificate-path]
[--byo-ca-key key-path]
[--cert-dir certificate-directory]
[--cert-request-common-name common_name]
[--cert-request-country country]
[--cert-request-locality locality]
[--cert-request-organization organization]
[--cert-request-organization-unit organization-unit]
[--cert-request-state state]
[{-h|--help}]
{-n|--nodes} nodes
[--one-cert]
```

The `--cert-dir` option sets the location where the certificates are to be saved. We recommend to use the the following format for the directory:

```
$HOME/.olcne/certificates/api_server_hostname:port
```

The `api_server_hostname` is the hostname of the node where the Platform API Server is installed, and the `port` is the port to access the Platform API Server (the default is `8091`). This is the default directory for the Platform API Server keys, and where the Platform CLI checks for access the Platform API Server without any extra configuration.

> **Important:**
>
> This is the path to specify the location of the certificates for the Platform CLI to access the Platform API Server when you create an environment using the `olcnectl environment create` command.

The `--nodes` option must be set to the hostname and IP address of the Platform API Server, as shown:

```
--nodes api_server_hostname,api_server_ip_address
```

Use the `--one-cert` option to save the certificates for the hostname and for the IP address to a single file.

You can use the same CA certificate and private key you used to generate the Kubernetes node certificates by using the `--byo-ca-cert` and `--byo-ca-key` options.

For example:

```
olcnectl certificates generate \
--nodes operator.example.com,127.0.0.1 \
--cert-dir $HOME/.olcne/certificates/operator.example.com:8091/  \
--byo-ca-cert $HOME/certificates/ca/ca.cert \
--byo-ca-key $HOME/certificates/ca/ca.key \
--one-cert
```

In this example, the certificate files (`node.cert`, `node.csr`, and `node.key`) are created and saved in the directory:

```
$HOME/.olcne/certificates/operator.example.com:8091/
```

Copy the CA Certificate you used to generate the keys to this directory. For example:

```
cp $HOME/certificates/ca/ca.cert $HOME/.olcne/certificates/
operator.example.com:8091/
```

The Platform CLI is set up to connect securely to the Platform API Server.

Setting up Certificates for the externalIPs Kubernetes Service

When you deploy Kubernetes, a service is deployed to the cluster that controls access to `externalIPs` in Kubernetes services. The service is named `externalip-validation-webhook-service` and runs in the `externalip-validation-system` namespace. This Kubernetes service requires X.509 certificates be set up before deploying Kubernetes. You can use Vault to generate the certificates, or use existing certificates for this purpose. You can also generate certificates using the Platform CLI. The certificates must be available on the operator node.

The examples in this book use the `/etc/olcne/certificates/restrict_external_ip/` directory for these certificates.

Setting up Vault Certificates

You can use Vault to generate a certificates for the `externalIPs` Kubernetes service. The Vault instance must be configured in the same way as described in Setting up Vault Authentication.

You need to generate certificates for two nodes, named:

```
externalip-validation-webhook-service.externalip-validation-system.svc
```

```
externalip-validation-webhook-service.externalip-validation-
system.svc.cluster.local
```

The certificate information must be generated in PEM format.

For example:

```
vault write olcne_pki_intermediary/issue/olcne \
    alt_names=externalip-validation-webhook-service.externalip-
validation-system.svc,\
externalip-validation-webhook-service.externalip-validation-
system.svc.cluster.local \
    format=pem_bundle
```

The output is displayed. Look for the section that starts with `certificate`. This section contains the certificates for the node names (set with the `alt_names` option). Save the output in this section to a file named `node.cert`. The file looks similar to:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAymg8uHy+mpwlelCyC4WrnfLwUmJ5vZmSos85QnIlZvyycUPK
...
X3c8LNaJDfQx1wKfTc/c0czBhHYxgwfau0G6wjqScZesPi2xY0xyslE=
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIID2TCCAsGgAwIBAgIUZ/M/D7bAjhyGx7DivsjBb9oeLhAwDQYJKoZIhvcNAQEL
...
9bRwnen+JrxUn4GV59GtsTiqzY6R2OKPm+zLl8E=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIDnDCCAoSgAwIBAgIUMapl4aWnBXE/02qTW0zOZ9aQVGgwDQYJKoZIhvcNAQEL
...
kV8w2xVXXAehp7cg0BakVA==
-----END CERTIFICATE-----
```

Look for the section that starts with `issuing_ca`. This section contains the CA certificate. Save the output in this section to a file named `ca.cert`. The file looks similar to:

```
-----BEGIN CERTIFICATE-----
MIIDnDCCAoSgAwIBAgIUMapl4aWnBXE/02qTW0zOZ9aQVGgwDQYJKoZIhvcNAQEL
...
kV8w2xVXXAehp7cg0BakVA==
-----END CERTIFICATE-----
```

Look for the section that starts with `private_key`. This section contains the private key for the node certificates. Save the output in this section to a file named `node.key`. The file looks similar to:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAymg8uHy+mpwlelCyC4WrnfLwUmJ5vZmSos85QnIlZvyycUPK
...
X3c8LNaJDfQx1wKfTc/c0czBhHYxgwfau0G6wjqScZesPi2xY0xyslE=
-----END RSA PRIVATE KEY-----
```

Copy the three files (`node.cert`, `ca.cert` and `node.key`) to the operator node and set the ownership of the files as described in Setting up CA Certificates.

Setting up CA Certificates

If you're using existing certificates, copy them to a directory under `/etc/olcne/certificates/` on the operator node. For example:

- **CA Certificate**: `/etc/olcne/certificates/restrict_external_ip/ca.cert`
- **Node Key**: `/etc/olcne/certificates/restrict_external_ip/node.key`
- **Node Certificate**: `/etc/olcne/certificates/restrict_external_ip/node.cert`

Copy these certificates to a different location on the operator node than the certificates and keys used for the Kubernetes nodes as set up in Setting up Certificates for Kubernetes Nodes. This makes sure you don't overwrite those certificates and keys. You need to generate certificates for two nodes, named:

```
externalip-validation-webhook-service.externalip-validation-system.svc
```

```
externalip-validation-webhook-service.externalip-validation-
system.svc.cluster.local
```

Save the certificates for these two nodes as a single file named `node.cert`.

Ensure the permissions of the directory where the certificates are saved can be read by the user on the operator node that you intend to use to run the `olcnectl` commands to install Kubernetes.

```
sudo chown -R username:username /etc/olcne/certificates/restrict_external_ip/
```

Setting up Private CA Certificates

You can use the Platform CLI to generate the certificates.

Generate Certificates for ExternalIPs Service

Use the Platform CLI to generate signed certificates for the Kubernetes ExternalIPs service.

On the operator node, use the `olcnectl certificates generate` command to generate certificates for the Kubernetes ExternalIPs service. The syntax to use is:

```
olcnectl certificates generate
[--byo-ca-cert certificate-path]
[--byo-ca-key key-path]
[--cert-dir certificate-directory]
[--cert-request-common-name common_name]
[--cert-request-country country]
[--cert-request-locality locality]
[--cert-request-organization organization]
```

```
[--cert-request-organization-unit organization-unit]
[--cert-request-state state]
[{-h|--help}]
{-n|--nodes} nodes
[--one-cert]
```

The `--cert-dir` option sets the location where the certificates are to be saved. We recommend you use the directory:

```
$HOME/certificates/restrict_external_ip/
```

The directory you use must have read and write permissions by the user you intend to use use to run the `olcnectl` commands to install Kubernetes.

> **Important:**
>
> This is the path to specify the location of the ExternalIPs Kubernetes service certificates when you create the Kubernetes module using the `olcnectl module create` command.

The `--nodes` option must be set to the name of the Kubernetes service, as shown:

```
--nodes externalip-validation-webhook-service.externalip-validation-
system.svc,externalip-validation-webhook-service.externalip-validation-
system.svc.cluster.local
```

Use the `--one-cert` option to save the certificates for the two service names to a single file.

You can use the same CA certificate and private key you used to generate the Kubernetes node certificates by using the `--byo-ca-cert` and `--byo-ca-key` options.

For example:

```
olcnectl certificates generate \
--nodes externalip-validation-webhook-service.externalip-validation-
system.svc,\
externalip-validation-webhook-service.externalip-validation-
system.svc.cluster.local \
--cert-dir $HOME/certificates/restrict_external_ip/ \
--byo-ca-cert $HOME/certificates/ca/ca.cert \
--byo-ca-key $HOME/certificates/ca/ca.key \
--one-cert
```

In this example, the certificates are created and saved in the directory:

```
$HOME/certificates/restrict_external_ip/
```

Copy the CA certificate you used to generate the certificate into to this directory. If you used all the defaults and suggested paths, the following command copies the file:

```
cp $HOME/certificates/ca/ca.cert $HOME/certificates/
restrict_external_ip/
```

The certificates are set up for the Kubernetes ExternalIPs service.

Starting the Platform API Server and Platform Agent Services

This section discusses using certificates to set up secure communication between the Platform API Server and the Platform Agent on nodes in the cluster. You can set up secure communication using certificates managed by Vault, or using certificates copied to each node. You must configure the Platform API Server and the Platform Agent to use the certificates when you start the services.

For information on setting up the certificates with Vault, see Setting up Certificates for Kubernetes Nodes.

For information on creating a private CA to sign certificates that can be used during testing, see Setting up Private CA Certificates.

Starting the Services Using Vault

This section shows you how to set up the Platform API Server and Platform Agent services to use certificates managed by Vault.

To set up and start the services using Vault:

1. On the operator node, use the `/etc/olcne/bootstrap-olcne.sh` script to configure the Platform API Server to retrieve and use a Vault certificate. Use the `bootstrap-olcne.sh --help` command for a list of options for this script. For example:

   ```
   sudo /etc/olcne/bootstrap-olcne.sh \
   --secret-manager-type vault \
   --vault-token s.3QKNuRoTqLbjXaGBOmO6Psjh \
   --vault-address https://192.0.2.20:8200 \
   --force-download-certs \
   --olcne-component api-server
   ```

   The certificates are generated and downloaded from Vault.

   By default, the certificates are saved to the `/etc/olcne/certificates/` directory. You can optionally specify a path for the certificates, for example, by including the following options in the `bootstrap-olcne.sh` command:

   ```
   --olcne-ca-path /path/ca.cert \
   --olcne-node-cert-path /path/node.cert \
   --olcne-node-key-path /path/node.key \
   ```

   The Platform API Server is configured to use the certificates, and started. You can confirm the service is running using:

   ```
   systemctl status olcne-api-server.service
   ```

2. On each Kubernetes node, use the `/etc/olcne/bootstrap-olcne.sh` script to configure the Platform Agent to retrieve and use a certificate. For example:

   ```
   sudo /etc/olcne/bootstrap-olcne.sh \
   --secret-manager-type vault \
   ```

```
--vault-token s.3QKNuRoTqLbjXaGBOmO6Psjh \
--vault-address https://192.0.2.20:8200 \
--force-download-certs \
--olcne-component agent
```

The certificates are generated and downloaded from Vault.

By default, the certificates are saved to the `/etc/olcne/certificates/` directory. You can optionally specify a path for the certificates, for example, by including the following options in the `bootstrap-olcne.sh` command:

```
--olcne-ca-path /path/ca.cert \
--olcne-node-cert-path /path/node.cert \
--olcne-node-key-path /path/node.key \
```

The Platform Agent is configured to use the certificates, and started. You can confirm the service is running using:

```
systemctl status olcne-agent.service
```

Starting the Services Using Certificates

This section shows you how to set up the Platform API Server and Platform Agent services to use certificates which have been copied to each node. This example assumes the certificates are available on all nodes in the `/etc/olcne/certificates/` directory.

To set up and start the services using certificates:

1. On the operator node, use the `/etc/olcne/bootstrap-olcne.sh` script to configure the Platform API Server to use the certificates. Use the `bootstrap-olcne.sh --help` command for a list of options for this script. For example:

   ```
   sudo /etc/olcne/bootstrap-olcne.sh \
   --secret-manager-type file \
   --olcne-component api-server
   ```

   If the certificates for Kubernetes nodes are in a directory other than `/etc/olcne/certificates/` (the default), include the location of the certificates. The `--olcne-node-cert-path`, `--olcne-ca-path`, and `--olcne-node-key-path` options set the location of the certificate files. For example:

   ```
   --olcne-node-cert-path /etc/olcne/configs/certificates/production/
   node.cert \
   --olcne-ca-path /etc/olcne/configs/certificates/production/ca.cert \
   --olcne-node-key-path /etc/olcne/configs/certificates/production/
   node.key \
   ```

The Platform API Server is configured to use the certificates, and started. You can confirm the service is running using:

```
systemctl status olcne-api-server.service
```

2. On each Kubernetes node, use the `/etc/olcne/bootstrap-olcne.sh` script to configure the Platform Agent to use the certificates. For example:

```
sudo /etc/olcne/bootstrap-olcne.sh \
--secret-manager-type file \
--olcne-component agent
```

If the certificates for Kubernetes nodes are in a directory other than `/etc/olcne/certificates/` (the default), include the location of the certificates. The `--olcne-node-cert-path`, `--olcne-ca-path`, and `--olcne-node-key-path` options set the location of the certificate files. For example:

```
--olcne-node-cert-path /etc/olcne/configs/certificates/production/
node.cert \
--olcne-ca-path /etc/olcne/configs/certificates/production/ca.cert \
--olcne-node-key-path /etc/olcne/configs/certificates/production/node.key
\
```

The Platform Agent is configured to use the certificates, and started. You can confirm the service is running using:

```
systemctl status olcne-agent.service
```

4
Creating an Environment

The first step to creating a Kubernetes cluster is to create an environment. You can create many environments, with each environment containing many modules. Naming each environment and module makes it easier to manage the deployed components of Oracle Cloud Native Environment.

> **Important:**
>
> Don't use the same node in more than one environment.

Use the `olcnectl environment create` command on the operator node to create an environment. For more information on the syntax for the `olcnectl environment create` command, see Platform Command-Line Interface.

> **Tip:**
>
> You can also use a configuration file to create an environment. The configuration file is a YAML file that contains the information about the environments and modules you want to deploy. Using a configuration file reduces the information you need to provide with `olcnectl` commands. For information on creating and using a configuration file, see Platform Command-Line Interface.

This section shows you how to create an environment using Vault, and using certificates copied to the file system on each node. For information on setting up X.509 certificates, see Setting up Certificates for Kubernetes Nodes.

Creating an Environment using Certificates Managed by Vault

This section shows you how to create an environment using Vault to provide and manage the certificates.

On the operator node, use the `olcnectl environment create` command to create an environment. For example, to create an environment named `myenvironment` using certificates generated from a Vault instance at `https://192.0.2.20:8200`:

```
olcnectl environment create \
--api-server 127.0.0.1:8091 \
--environment-name myenvironment \
--secret-manager-type vault \
--vault-token s.3QKNuRoTqLbjXaGBOmO6Psjh \
--vault-address https://192.0.2.20:8200 \
--update-config
```

The `--api-server` option sets the location of the Platform API Server service. In this example, the Platform API Server is running on the operator node (the localhost) and listening on port `8091`.

The `--environment-name` option sets the name of the environment, which in this example is `myenvironment`.

The `--secret-manager-type` option sets the certificate manager to Vault.

Replace `--vault-token` with the token to access Vault.

Replace `--vault-address` with the location of the Vault instance.

The `--update-config` option writes information about the environment to a local configuration file at `$HOME/.olcne/olcne.conf`, and this configuration is used for future calls to the Platform API Server. If you use this option, you don't need to specify the Platform API Server (using the `--api-server` option) in future `olcnectl` commands. For more information on setting the Platform API Server see Platform Command-Line Interface.

By default, the certificate generated by Vault is saved to `$HOME/.olcne/certificates/`*`environment_name`*`/`. To specify a different location to save the certificate, use the `--olcne-node-cert-path`, `--olcne-ca-path`, and `--olcne-node-key-path` options. For example, add the following options to the `olcnectl environment create` command:

```
--olcne-node-cert-path /path/node.cert \
--olcne-ca-path /path/ca.cert \
--olcne-node-key-path /path/node.key
```

Creating an Environment using Certificates

This section shows you how to create an environment using certificates, copied to each Kubernetes node. This example assumes the certificates are available on all nodes in the `/etc/olcne/certificates/` directory.

On the operator node, create the environment using the `olcnectl environment create` command. For example:

```
olcnectl environment create \
--api-server 127.0.0.1:8091 \
--environment-name myenvironment \
--secret-manager-type file \
--olcne-node-cert-path /etc/olcne/certificates/node.cert \
--olcne-ca-path /etc/olcne/certificates/ca.cert \
--olcne-node-key-path /etc/olcne/certificates/node.key \
--update-config
```

If you created certificates for the Platform CLI to communicate the with Platform API Server as shown in Generate Certificates for the Platform CLI to the Platform API Server, you don't need to specify the location of the certificate information. The key information is stored in:

`$HOME/.olcne/certificates/`*`api_server_hostname`*`:`*`port`*

If you followed that procedure, you can create an environment without the key information, for example:

```
olcnectl environment create \
--api-server 127.0.0.1:8091 \
--environment-name myenvironment \
--secret-manager-type file \
--update-config
```

The `--api-server` option sets the location of the Platform API Server service. In this example, the Platform API Server is running on the operator node (the localhost) and listening on port `8091`.

The `--environment-name` option sets the name of the environment, which in this example is `myenvironment`.

The `--secret-manager-type` option sets the certificate manager to use file-based certificates.

The `--olcne-node-cert-path`, `--olcne-ca-path`, and `--olcne-node-key-path` options set the location of the certificate files.

You can optionally set the location for the certificate files using environment variables as the Platform CLI uses these if they're set. The following environment variables map to the `olcnectl environment create` command options:

Table 4-1 Certificate Options

Command Option	Environment Variable	Purpose
`--olcne-node-cert-path`	`$OLCNE_SM_CERT_PATH`	The path to the node certificate.
`--olcne-ca-path`	`$OLCNE_SM_CA_PATH`	The path to the Certificate Authority certificate.
`--olcne-node-key-path`	`$OLCNE_SM_KEY_PATH`	The path to the key for the node's certificate.

For example, to set the certificate information using environment variables for the same environment, you could use:

```
export OLCNE_SM_CA_PATH=/etc/olcne/certificates//ca.cert
export OLCNE_SM_CERT_PATH=/etc/olcne/certificates/node.cert
export OLCNE_SM_KEY_PATH=/etc/olcne/certificates/node.key

olcnectl environment create \
--api-server 127.0.0.1:8091 \
--environment-name myenvironment \
--secret-manager-type file \
--update-config
```

The `--update-config` option writes information about the environment to a local configuration file at `$HOME/.olcne/olcne.conf`, and this configuration is used for future calls to the Platform API Server. If you use this option, you don't need to specify the Platform API Server (using the `--api-server` option) in future `olcnectl` commands. For more information on setting the Platform API Server see Platform Command-Line Interface.

5

Installing Modules

After you create an environment, you can add any modules you want to the environment.

Kubernetes Module

A base installation requires a Kubernetes module which is used to create a Kubernetes cluster. For information on creating and installing a Kubernetes module, see Kubernetes Module.

Calico Module

When you have created and installed a Kubernetes module, you can optionally install the Calico module. Calico can be used to configure the Kubernetes CNI to manage pod network traffic. For information on creating and installing a Calico module, see Calico Module.

Multus Module

When you have created and installed a Kubernetes module, you can optionally install the Multus module. Multus can be used to create a network bridge between either Flannel or Calico. For information on creating and installing a Multus module, see Multus Module.

Oracle Cloud Infrastructure Cloud Controller Manager Module

When you have created and installed a Kubernetes module, you can optionally install the Oracle Cloud Infrastructure Cloud Controller Manager module to set up access to Oracle Cloud Infrastructure storage and application load balancers. This lets you use Oracle Cloud Infrastructure storage to provide persistent storage for Kubernetes applications. This also lets you create load balancers for Kubernetes applications so they can be accessed externally, from outside the cluster.

For information on installing the Oracle Cloud Infrastructure Cloud Controller Manager module, see Oracle Cloud Infrastructure Cloud Controller Manager Module.

MetalLB Module

When you have created and installed a Kubernetes module, you can optionally install the MetalLB module. MetalLB is a network load balancer for Kubernetes applications running on bare metal hosts. MetalLB lets you use Kubernetes LoadBalancer services, which traditionally use a cloud provider network load balancer, in a bare metal environment. For information on installing the MetalLB module, see MetalLB Module.

Rook Module

When you have created and installed a Kubernetes module, you can optionally install the Rook module to set up access to Ceph storage. This lets you use a Ceph cluster to provide persistent storage for Kubernetes applications. For information on installing the Rook module, see Rook Module.

KubeVirt Module

When you have created and installed a Kubernetes module, you can optionally install the KubeVirt module to create and manage virtual machines. For information on installing the KubeVirt module, see KubeVirt Module.

Operator Lifecycle Manager Module

When you have created and installed a Kubernetes module, you can optionally install the Operator Lifecycle Manager module to manage the installation and lifecycle management of operators in a Kubernetes cluster. For information on installing the Operator Lifecycle Manager module, see Operator Lifecycle Manager Module.

Istio Module

When you have created and installed a Kubernetes module, you can optionally install a service mesh using the Istio module. For information on installing the Istio module to create a service mesh, see Istio Module.

6
Configuring Services

This chapter contains information about any configuration options for Oracle Cloud Native Environment, including configuring the Platform API Server and Platform Agent services.

Configuring the Platform API Server

The Platform API Server runs as a Systemd service, named `olcne-api-server`. You can get logs for this service using:

```
sudo journalctl -u olcne-api-server
```

By default, the service logs information level messages. You can change this to log debug level messages using the `OLCNE_DEBUG` environment variable or a Systemd drop in file.

If the environment variable is set to `OLCNE_DEBUG=0`, the logging level is set to information messages. Setting `OLCNE_DEBUG=1` sets the logging level to debug messages, for example:

```
export OLCNE_DEBUG=1
```

You can also set this as a Systemd drop in file. Edit the `/etc/systemd/system/olcne-api-server.service.d/10-auth.conf` file and change the `OLCNE_DEBUG` entry. For example:

```
[Service]
Environment="VAULT_SKIP_VERIFY=false"
Environment="OLCNE_DEBUG=1"
Environment="OLCNE_AGENT_ARGS= --secret-manager-type 'file' "
Environment="OLCNE_TLS=  "
ExecStart=
ExecStart=/usr/libexec/olcne-api-server -i /etc/olcne/
modules $OLCNE_AGENT_ARGS $OLCNE_TLS
RestartSec=5
```

You can change some Platform API Server settings by editing the Systemd service unit file so that the binary is invoked to use extra options.

`olcne-api-server` options:

- `[-p|--port]` *`port_number`*

  Specifies the port that the Platform API Server binds to. Defaults to `8091` if unspecified.

- `[-i|--installables]` *`installables_path`*

  Specifies the path to the directory of installable modules. Defaults to `/etc/olcne/modules`.

- `[-x|--insecure]`

Allows the gRPC server to accept clients that don't securely establish their identity.

To reconfigure the Platform API Server to use any of these options, you can edit the Systemd unit file at `/usr/lib/systemd/system/olcne-api-server.service` and append the option to the `ExecStart` line. For example:

```
[Unit]
Description=Platform API Server for Oracle Cloud Native Environments
Wants=network.target
After=network.target

[Service]
ExecStart=/usr/libexec/olcne-api-server -i /etc/olcne/modules --port
9083
WorkingDirectory=/var/olcne
User=olcne
Group=olcne
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

If you edit the Systemd unit or drop in file, you must run the following commands for the changes to take effect:

```
sudo systemctl daemon-reload
sudo systemctl restart olcne-api-server.service
```

> **Note:**
>
> If you change the port value for this service, take this into account for all other instructions provided in the documentation.

Configuring the Platform Agent

The Platform Agent runs as a Systemd service, named `olcne-agent`. You can get logs for this service using:

```
sudo journalctl -u olcne-agent
```

By default, the service logs information level messages. You can change this to log debug level messages using the `OLCNE_DEBUG` environment variable or a Systemd drop in file.

If the environment variable is set to `OLCNE_DEBUG=0`, the logging level is set to information messages. Setting `OLCNE_DEBUG=1` sets the logging level to debug messages, for example:

```
export OLCNE_DEBUG=1
```

You can also set this as a Systemd drop in file. Edit the `/etc/systemd/system/olcne-agent.service.d/10-auth.conf` file and change the `OLCNE_DEBUG` entry. For example:

```
[Service]
Environment="VAULT_SKIP_VERIFY=false"
Environment="OLCNE_DEBUG=1"
Environment="OLCNE_AGENT_ARGS= --secret-manager-type 'file' "
Environment="OLCNE_TLS=  "
ExecStart=
ExecStart=/usr/libexec/olcne-agent  $OLCNE_AGENT_ARGS $OLCNE_TLS
RestartSec=5
```

You can change some Platform Agent settings by editing the Systemd service unit file so that the binary is invoked to use extra options.

`olcne-agent` options:

- `[-p|--port]` *`port-number`*

  Specifies the port that the Platform Agent service binds to. Defaults to `8090` if unspecified.

- `[-x|--insecure]`

  Allows the gRPC server to accept clients that don't securely establish their identity.

To reconfigure the Platform Agent to use any of these options, you can edit the Systemd unit file at `/usr/lib/systemd/system/olcne-agent.service` and append the option to the `ExecStart` line.

If you edit the Systemd unit or drop in file, you must run the following commands for the changes to take effect:

```
sudo systemctl daemon-reload
sudo systemctl restart olcne-agent.service
```

> **Note:**
>
> If you change the port value for this service, take this into account for all other instructions provided in the documentation.