OpenShift Container Platform 4.5

Installing

Installing and configuring OpenShift Container Platform clusters

Last Updated: 2020-11-20

OpenShift Container Platform 4.5 Installing

Installing and configuring OpenShift Container Platform clusters

Legal Notice

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This document provides information about installing OpenShift Container Platform and details about some configuration processes.

Table of Contents

CHAPTER 1. TROUBLESHOOTING INSTALLATION ISSUES

To assist in troubleshooting a failed OpenShift Container Platform installation, you can gather logs from the bootstrap and control plane, or master, machines. You can also get debug information from the installation program.

1.1. PREREQUISITES

- You attempted to install an OpenShift Container Platform cluster, and installation failed.

1.2. GATHERING LOGS FROM A FAILED INSTALLATION

If you gave an SSH key to your installation program, you can gather data about your failed installation.

NOTE

You use a different command to gather logs about an unsuccessful installation than to gather logs from a running cluster. If you must gather logs from a running cluster, use the **oc adm must-gather** command.

Prerequisites

- Your OpenShift Container Platform installation failed before the bootstrap process finished. The bootstrap node is running and accessible through SSH.
- The **ssh-agent** process is active on your computer, and you provided the same SSH key to both the **ssh-agent** process and the installation program.
- If you tried to install a cluster on infrastructure that you provisioned, you must have the fully qualified domain names of the bootstrap and master nodes.

Procedure

1. Generate the commands that are required to obtain the installation logs from the bootstrap and control plane machines:
   - If you used installer-provisioned infrastructure, run the following command:

     ```
     $ ./openshift-install gather bootstrap --dir=<installation_directory> 1
     ```

     1 **installation_directory** is the directory you specified when you ran **./openshift-install create cluster**. This directory contains the OpenShift Container Platform definition files that the installation program creates.

     For installer-provisioned infrastructure, the installation program stores information about the cluster, so you do not specify the host names or IP addresses

   - If you used infrastructure that you provisioned yourself, run the following command:

     ```
     $ ./openshift-install gather bootstrap --dir=<installation_directory> \ 1
       --bootstrap <bootstrap_address> \ 2
       --master <master_1_address> \ 3
     ```

```
--master <master_2_address> \ 4
--master <master_3_address>" 5
```

1. For **installation_directory**, specify the same directory you specified when you ran **./openshift-install create cluster**. This directory contains the OpenShift Container Platform definition files that the installation program creates.
2. **<bootstrap_address>** is the fully qualified domain name or IP address of the cluster's bootstrap machine.
3. 4. 5. For each control plane, or master, machine in your cluster, replace **<master_*_address>** with its fully qualified domain name or IP address.

NOTE

A default cluster contains three control plane machines. List all of your control plane machines as shown, no matter how many your cluster uses.

Example output

```
INFO Pulling debug logs from the bootstrap machine
INFO Bootstrap gather logs captured here "<installation_directory>/log-bundle-
<timestamp>.tar.gz"
```

If you open a Red Hat support case about your installation failure, include the compressed logs in the case.

1.3. MANUALLY GATHERING LOGS WITH SSH ACCESS TO YOUR HOST(S)

Manually gather logs in situations where **must-gather** or automated collection methods do not work.

Prerequisites

- You must have SSH access to your host(s).

Procedure

1. Collect the **bootkube.service** service logs from the bootstrap host using the **journalctl** command by running:

```
$ journalctl -b -f -u bootkube.service
```

2. Collect the bootstrap host's container logs using the Podman logs. This is shown as a loop to get all of the container logs from the host:

```
$ for pod in $(sudo podman ps -a -q); do sudo podman logs $pod; done
```

3. Alternatively, collect the host's container logs using the **tail** command by running:

```
# tail -f /var/lib/containers/storage/overlay-containers/*/userdata/ctr.log
```

4. Collect the **kubelet.service** and **crio.service** service logs from the master and worker hosts using the **journalctl** command by running:

```
$ journalctl -b -f -u kubelet.service -u crio.service
```

5. Collect the master and worker host container logs using the **tail** command by running:

```
$ sudo tail -f /var/log/containers/*
```

1.4. MANUALLY GATHERING LOGS WITHOUT SSH ACCESS TO YOUR HOST(S)

Manually gather logs in situations where **must-gather** or automated collection methods do not work.

If you do not have SSH access to your node, you can access the systems journal to investigate what is happening on your host.

Prerequisites

- Your OpenShift Container Platform installation must be complete.
- Your API service is still functional.
- You have system administrator privileges.

Procedure

1. Access **journald** unit logs under **/var/log** by running:

```
$ oc adm node-logs --role=master -u kubelet
```

2. Access host file paths under **/var/log** by running:

```
$ oc adm node-logs --role=master --path=openshift-apiserver
```

1.5. GETTING DEBUG INFORMATION FROM THE INSTALLATION PROGRAM

You can use any of the following actions to get debug information from the installation program.

- Look at debug messages from a past installation in the hidden **.openshift_install.log** file. For example, enter:

```
$ cat ~/<installation_directory>/.openshift_install.log 1
```

1 For **installation_directory**, specify the same directory you specified when you ran **./openshift-install create cluster**.

- Re-run the installation program with **--log-level=debug**:

```
$ ./openshift-install create cluster --dir=<installation_directory> --log-level=debug 1
```

-

1 For **installation_directory**, specify the same directory you specified when you ran **./openshift-install create cluster**.

CHAPTER 2. SUPPORT FOR FIPS CRYPTOGRAPHY

Starting with version 4.3, you can install an OpenShift Container Platform cluster that uses FIPS Validated / Modules in Process cryptographic libraries.

For the Red Hat Enterprise Linux CoreOS (RHCOS) machines in your cluster, this change is applied when the machines are deployed based on the status of an option in the **install-config.yaml** file, which governs the cluster options that a user can change during cluster deployment. With Red Hat Enterprise Linux machines, you must enable FIPS mode when you install the operating system on the machines that you plan to use as worker machines. These configuration methods ensure that your cluster meet the requirements of a FIPS compliance audit: only FIPS Validated / Modules in Process cryptography packages are enabled before the initial system boot.

Because FIPS must be enabled before the operating system that your cluster uses boots for the first time, you cannot enable FIPS after you deploy a cluster.

2.1. FIPS VALIDATION IN OPENSHIFT CONTAINER PLATFORM

OpenShift Container Platform uses certain FIPS Validated / Modules in Process modules within Red Hat Enterprise Linux (RHEL) and RHCOS for the operating system components that it uses. See RHEL7 core crypto components. For example, when users SSH into OpenShift Container Platform clusters and containers, those connections are properly encrypted.

OpenShift Container Platform components are written in Go and built with Red Hat's golang compiler. When you enable FIPS mode for your cluster, all OpenShift Container Platform components that require cryptographic signing call RHEL and RHCOS cryptographic libraries.

Table 2.1. FIPS mode attributes and limitations in OpenShift Container Platform 4.5

Attributes	Limitations
FIPS support in RHEL 7 operating systems.	The FIPS implementation does not offer a single function that both computes hash functions and validates the keys that are based on that hash. This limitation will continue to be evaluated and improved in future OpenShift Container Platform releases.
FIPS support in CRI-O runtimes.	
FIPS support in OpenShift Container Platform services.	
FIPS Validated / Modules in Process cryptographic module and algorithms that are obtained from RHEL 7 and RHCOS binaries and images.	
Use of FIPS compatible golang compiler.	TLS FIPS support is not complete but is planned for future OpenShift Container Platform releases.

2.2. FIPS SUPPORT IN COMPONENTS THAT THE CLUSTER USES

Although the OpenShift Container Platform cluster itself uses FIPS Validated / Modules in Process modules, ensure that the systems that support your OpenShift Container Platform cluster use FIPS Validated / Modules in Process modules for cryptography.

2.2.1. etcd

To ensure that the secrets that are stored in etcd use FIPS Validated / Modules in Process encryption, boot the node in FIPS mode. After you install the cluster in FIPS mode, you can encrypt the etcd data by using the FIPS-approved **aes cbc** cryptographic algorithm.

2.2.2. Storage

For local storage, use RHEL-provided disk encryption or Container Native Storage that uses RHEL-provided disk encryption. By storing all data in volumes that use RHEL-provided disk encryption and enabling FIPS mode for your cluster, both data at rest and data in motion, or network data, are protected by FIPS Validated / Modules in Process encryption. You can configure your cluster to encrypt the root filesystem of each node, as described in Customizing nodes.

2.2.3. Runtimes

To ensure that containers know that they are running on a host that is using FIPS Validated / Modules in Process cryptography modules, use CRI-O to manage your runtimes. CRI-O supports FIPS-Mode, in that it configures the containers to know that they are running in FIPS mode.

2.3. INSTALLING A CLUSTER IN FIPS MODE

To install a cluster in FIPS mode, follow the instructions to install a customized cluster on your preferred infrastructure. Ensure that you set **fips: true** in the **install-config.yaml** file before you deploy your cluster.

- Amazon Web Services
- Microsoft Azure
- Bare metal
- Google Cloud Platform
- Red Hat OpenStack Platform (RHOSP)
- VMware vSphere

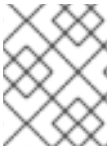
To apply **AES CBC** encryption to your etcd data store, follow the Encrypting etcd data process after you install your cluster.

If you add RHEL nodes to your cluster, ensure that you enable FIPS mode on the machines before their initial boot. See Adding RHEL compute machines to an OpenShift Container Platform cluster and Enabling FIPS Mode in the RHEL 7 documentation.

CHAPTER 3. INSTALLATION CONFIGURATION

3.1. INSTALLATION METHODS FOR DIFFERENT PLATFORMS

You can perform different types of installations on different platforms.



NOTE

Not all installation options are currently available for all platforms, as shown in the following tables.

Table 3.1. Installer-provisioned infrastructure options

	AWS	Azure	GCP	OpenStack	RHV	Bare metal	vSphere	IBM Z
Default	X	X	X		X			
Custom	X	X	X	X	X			
Network Operator	X	X	X					
Private clusters	X	X	X					
Existing virtual private networks	X	X	X					

Table 3.2. User-provisioned infrastructure options

	AWS	Azure	GCP	OpenStack	RHV	Bare metal	vSphere	IBM Z
Custom	X	X	X	X		X	X	
Network Operator						X	X	
Restricted network	X		X			X	X	

3.2. CUSTOMIZING NODES

Although directly making changes to OpenShift Container Platform nodes is discouraged, there are times when it is necessary to implement a required low-level security, networking, or performance feature. Direct changes to OpenShift Container Platform nodes can be done by:

- Creating MachineConfigs that are included in manifest files to start up a cluster during **openshift-install**.
- Creating MachineConfigs that are passed to running OpenShift Container Platform nodes via the Machine Config Operator.

The following sections describe features that you might want to configure on your nodes in this way.

3.2.1. Adding day-1 kernel arguments

Although it is often preferable to modify kernel arguments as a day-2 activity, you might want to add kernel arguments to all master or worker nodes during initial cluster installation. Here are some reasons you might want to add kernel arguments during cluster installation so they take effect before the systems first boot up:

- You want to disable a feature, such as SELinux, so it has no impact on the systems when they first come up.
- You need to do some low-level network configuration before the systems start.

To add kernel arguments to master or worker nodes, you can create a **MachineConfig** object and inject that object into the set of manifest files used by Ignition during cluster setup.

For a listing of arguments you can pass to a RHEL 8 kernel at boot time, see Kernel.org kernel parameters. It is best to only add kernel arguments with this procedure if they are needed to complete the initial OpenShift Container Platform installation.

Procedure

1. Generate the Kubernetes manifests for the cluster:

```
$ ./openshift-install create manifests --dir=<installation_directory>
```

2. Decide if you want to add kernel arguments to worker or master nodes.
3. In the **openshift** directory, create a file (for example, **99-openshift-machineconfig-master-kargs.yaml**) to define a MachineConfig object to add the kernel settings. This example adds a **loglevel=7** kernel argument to master nodes:

```
$ cat << EOF > 99-openshift-machineconfig-master-kargs.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 99-openshift-machineconfig-master-kargs
spec:
```

```
  kernelArguments:
    - 'loglevel=7'
EOF
```

You can change **master** to **worker** to add kernel arguments to worker nodes instead. Create a separate YAML file to add to both master and worker nodes.

You can now continue on to create the cluster.

3.2.2. Adding kernel modules to nodes

For most common hardware, the Linux kernel includes the device driver modules needed to use that hardware when the computer starts up. For some hardware, however, modules are not available in Linux. Therefore, you must find a way to provide those modules to each host computer. This procedure describes how to do that for nodes in an OpenShift Container Platform cluster.

When a kernel module is first deployed by following these instructions, the module is made available for the current kernel. If a new kernel is installed, the kmods-via-containers software will rebuild and deploy the module so a compatible version of that module is available with the new kernel.

The way that this feature is able to keep the module up to date on each node is by:

- Adding a systemd service to each node that starts at boot time to detect if a new kernel has been installed and
- If a new kernel is detected, the service rebuilds the module and installs it to the kernel

For information on the software needed for this procedure, see the kmods-via-containers github site.

A few important issues to keep in mind:

- This procedure is Technology Preview.
- Software tools and examples are not yet available in official RPM form and can only be obtained for now from unofficial **github.com** sites noted in the procedure.
- Third-party kernel modules you might add through these procedures are not supported by Red Hat.
- In this procedure, the software needed to build your kernel modules is deployed in a RHEL 8 container. Keep in mind that modules are rebuilt automatically on each node when that node gets a new kernel. For that reason, each node needs access to a **yum** repository that contains the kernel and related packages needed to rebuild the module. That content is best provided with a valid RHEL subscription.

3.2.2.1. Building and testing the kernel module container

Before deploying kernel modules to your OpenShift Container Platform cluster, you can test the process on a separate RHEL system. Gather the kernel module's source code, the KVC framework, and the kmod-via-containers software. Then build and test the module. To do that on a RHEL 8 system, do the following:

Procedure

1. Register a RHEL 8 system:

```
# subscription-manager register
```

2. Attach a subscription to the RHEL 8 system:

```
# subscription-manager attach --auto
```

3. Install software that is requried to build the software and container:

```
# yum install podman make git -y
```

4. Clone the **kmod-via-containers** repository:

   a. Create a folder for the repository:

   ```
   $ mkdir kmods; cd kmods
   ```

   b. Clone the repository:

   ```
   $ git clone https://github.com/kmods-via-containers/kmods-via-containers
   ```

5. Install a KVC framework instance on your RHEL 8 build host to test the module. This adds a **kmods-via-container** systemd service and loads it:

   a. Change to the **kmod-via-containers** directory:

   ```
   $ cd kmods-via-containers/
   ```

   b. Install the KVC framework instance:

   ```
   $ sudo make install
   ```

   c. Reload the systemd manager configuration:

   ```
   $ sudo systemctl daemon-reload
   ```

6. Get the kernel module source code. The source code might be used to build a third-party module that you do not have control over, but is supplied by others. You will need content similar to the content shown in the **kvc-simple-kmod** example that can be cloned to your system as follows:

```
$ cd .. ; git clone https://github.com/kmods-via-containers/kvc-simple-kmod
```

7. Edit the configuration file, **simple-kmod.conf** file, in this example, and change the name of the Dockerfile to **Dockerfile.rhel**:

   a. Change to the **kvc-simple-kmod** directory:

   ```
   $ cd kvc-simple-kmod
   ```

   b. Rename the Dockerfile:

   ```
   $ cat simple-kmod.conf
   ```

Example Dockerfile

```
KMOD_CONTAINER_BUILD_CONTEXT="https://github.com/kmods-via-containers/kvc-simple-kmod.git"
KMOD_CONTAINER_BUILD_FILE=Dockerfile.rhel
KMOD_SOFTWARE_VERSION=dd1a7d4
KMOD_NAMES="simple-kmod simple-procfs-kmod"
```

8. Create an instance of **kmods-via-containers@.service** for your kernel module, **simple-kmod** in this example:

```
$ sudo make install
```

9. Enable the **kmods-via-containers@.service** instance:

```
$ sudo kmods-via-containers build simple-kmod $(uname -r)
```

10. Enable and start the systemd service:

    a. Enable the service:

```
$ sudo systemctl enable kmods-via-containers@simple-kmod.service
```

    b. Start the service:

```
$ sudo systemctl start kmods-via-containers@simple-kmod.service
```

    c. Review the service status:

```
$ sudo systemctl status kmods-via-containers@simple-kmod.service
```

Example output

```
● kmods-via-containers@simple-kmod.service - Kmods Via Containers - simple-kmod
  Loaded: loaded (/etc/systemd/system/kmods-via-containers@.service;
        enabled; vendor preset: disabled)
  Active: active (exited) since Sun 2020-01-12 23:49:49 EST; 5s ago...
```

11. To confirm that the kernel modules are loaded, use the **lsmod** command to list the modules:

```
$ lsmod | grep simple_
```

Example output

```
simple_procfs_kmod     16384  0
simple_kmod            16384  0
```

12. Optional. Use other methods to check that the **simple-kmod** example is working:

- Look for a "Hello world" message in the kernel ring buffer with **dmesg**:

```
$ dmesg | grep 'Hello world'
```

Example output

```
[ 6420.761332] Hello world from simple_kmod.
```

- Check the value of **simple-procfs-kmod** in **/proc**:

```
$ sudo cat /proc/simple-procfs-kmod
```

Example output

```
simple-procfs-kmod number = 0
```

- Run the **spkut** command to get more information from the module:

```
$ sudo spkut 44
```

Example output

```
KVC: wrapper simple-kmod for 4.18.0-147.3.1.el8_1.x86_64
Running userspace wrapper using the kernel module container...
+ podman run -i --rm --privileged
  simple-kmod-dd1a7d4:4.18.0-147.3.1.el8_1.x86_64 spkut 44
simple-procfs-kmod number = 0
simple-procfs-kmod number = 44
```

Going forward, when the system boots this service will check if a new kernel is running. If there is a new kernel, the service builds a new version of the kernel module and then loads it. If the module is already built, it will just load it.

3.2.2.2. Provisioning a kernel module to OpenShift Container Platform

Depending on whether or not you must have the kernel module in place when OpenShift Container Platform cluster first boots, you can set up the kernel modules to be deployed in one of two ways:

- **Provision kernel modules at cluster install time (day-1)** You can create the content as a MachineConfig and provide it to **openshift-install** by including it with a set of manifest files.
- **Provision kernel modules via Machine Config Operator (day-2)** If you can wait until the cluster is up and running to add your kernel module, you can deploy the kernel module software via the Machine Config Operator (MCO).

In either case, each node needs to be able to get the kernel packages and related software packages at the time that a new kernel is detected. There are a few ways you can set up each node to be able to obtain that content.

- Provide RHEL entitlements to each node.
- Get RHEL entitlements from an existing RHEL host, from the **/etc/pki/entitlement** directory and copy them to the same location as the other files you provide when you build your Ignition config.

- Inside the Dockerfile, add pointers to a **yum** repository containing the kernel and other packages. This must include new kernel packages as they are needed to match newly installed kernels.

3.2.2.2.1. Provision kernel modules via a MachineConfig

By packaging kernel module software with a MachineConfig you can deliver that software to worker or master nodes at installation time or via the Machine Config Operator.

First create a base Ignition config that you would like to use. At installation time, the Ignition config will contain the ssh public key to add to the **authorized_keys** file for the **core** user on the cluster. To add the MachineConfig later via the MCO instead, the ssh public key is not required. For both type, the example simple-kmod service creates a systemd unit file, which requires a **kmods-via-containers@simple-kmod.service**.

NOTE

The systemd unit is a workaround for an upstream bug and makes sure that the **kmods-via-containers@simple-kmod.service** gets started on boot:

1. Register a RHEL 8 system:

```
# subscription-manager register
```

2. Attach a subscription to the RHEL 8 system:

```
# subscription-manager attach --auto
```

3. Install software needed to build the software:

```
# yum install podman make git -y
```

4. Create an Ignition config file that creates a systemd unit file:

   a. Create a directory to host the Ignition config file:

```
$ mkdir kmods; cd kmods
```

   b. Create the Ignition config file that creates a systemd unit file:

```
$ cat <<EOF > ./baseconfig.ign
{
  "ignition": { "version": "2.2.0" },
  "passwd": {
    "users": [
      {
        "name": "core",
        "groups": ["sudo"],
        "sshAuthorizedKeys": [
          "ssh-rsa AAAA"
        ]
      }
    ]
  },
```

```
  "systemd": {
    "units": [{
      "name": "require-kvc-simple-kmod.service",
      "enabled": true,
      "contents": "[Unit]\nRequires=kmods-via-containers@simple-
kmod.service\n[Service]\nType=oneshot\nExecStart=/usr/bin/true\n\n[Install]\nWantedBy=m
ulti-user.target"
    }]
  }
}
EOF
```

NOTE

You must add your public SSH key to the **baseconfig.ign** file to use the file during **openshift-install**. The public SSH key is not needed if you create the MachineConfig via the MCO.

5. Create a base MCO YAML snippet that uses the following configuration:

```
$ cat <<EOF > mc-base.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 10-kvc-simple-kmod
spec:
  config:
EOF
```

NOTE

The **mc-base.yaml** is set to deploy the kernel module on **worker** nodes. To deploy on master nodes, change the role from **worker** to **master**. To do both, you could repeat the whole procedure using different file names for the two types of deployments.

6. Get the **kmods-via-containers** software:

   a. Clone the **kmods-via-containers** repository:

   ```
   $ git clone https://github.com/kmods-via-containers/kmods-via-containers
   ```

   b. Clone the **kvc-simple-kmod** repository:

   ```
   $ git clone https://github.com/kmods-via-containers/kvc-simple-kmod
   ```

7. Get your module software. In this example, **kvc-simple-kmod** is used:

8. Create a fakeroot directory and populate it with files that you want to deliver via Ignition, using the repositories cloned earlier:

a. Create the directory:

```
$ FAKEROOT=$(mktemp -d)
```

b. Change to the **kmod-via-containers** directory:

```
$ cd kmods-via-containers
```

c. Install the KVC framework instance:

```
$ make install DESTDIR=${FAKEROOT}/usr/local CONFDIR=${FAKEROOT}/etc/
```

d. Change to the **kvc-simple-kmod** directory:

```
$ cd ../kvc-simple-kmod
```

e. Create the instance:

```
$ make install DESTDIR=${FAKEROOT}/usr/local CONFDIR=${FAKEROOT}/etc/
```

9. Get a tool called **filetranspiler** and dependent software:

```
$ cd .. ; sudo yum install -y python3
git clone https://github.com/ashcrow/filetranspiler.git
```

10. Generate a final MachineConfig YAML (**mc.yaml**) and have it include the base Ignition config, base MachineConfig, and the fakeroot directory with files you would like to deliver:

```
$ ./filetranspiler/filetranspile -i ./baseconfig.ign \
    -f ${FAKEROOT} --format=yaml --dereference-symlinks \
    | sed 's/^/     /' | (cat mc-base.yaml -) > 99-simple-kmod.yaml
```

11. If the cluster is not up yet, generate manifest files and add this file to the **openshift** directory. If the cluster is already running, apply the file as follows:

```
$ oc create -f 99-simple-kmod.yaml
```

Your nodes will start the **kmods-via-containers@simple-kmod.service** service and the kernel modules will be loaded.

12. To confirm that the kernel modules are loaded, you can log in to a node (using **oc debug node/<openshift-node>**, then **chroot /host**). To list the modules, use the **lsmod** command:

```
$ lsmod | grep simple_
```

Example output

```
simple_procfs_kmod     16384  0
simple_kmod            16384  0
```

3.2.3. Encrypting disks during installation

During OpenShift Container Platform installation, you can enable disk encryption on all master and worker nodes. This feature:

- Is available for installer-provisioned infrastructure and user provisioned infrastructure deployments
- Is supported on Red Hat Enterprise Linux CoreOS (RHCOS) systems only
- Sets up disk encryption during the manifest installation phase so all data written to disk, from first boot forward, is encrypted
- Encrypts data on the root filesystem only (**/dev/mapper/coreos-luks-root** on /)
- Requires no user intervention for providing passphrases
- Uses AES-256-CBC encryption
- Should be enabled for your cluster to support FIPS.

There are two different supported encryption modes:

- TPM v2: This is the preferred mode. TPM v2 stores passphrases in a secure cryptoprocessor. To implement TPM v2 disk encryption, create an Ignition config file as described below.
- Tang: To use Tang to encrypt your cluster, you need to use a Tang server. Clevis implements decryption on the client side. Tang encryption mode is only supported for bare metal installs.

Follow one of the two procedures to enable disk encryption for the nodes in your cluster.

3.2.3.1. Enabling TPM v2 disk encryption

Use this procedure to enable TPM v2 mode disk encryption during OpenShift Container Platform deployment.

Procedure

1. Check to see if TPM v2 encryption needs to be enabled in the BIOS on each node. This is required on most Dell systems. Check the manual for your computer.
2. Generate the Kubernetes manifests for the cluster:

```
$ ./openshift-install create manifests --dir=<installation_directory>
```

3. In the **openshift** directory, create master or worker files to encrypt disks for those nodes.
   - To create a worker file, run the following command:

```
$ cat << EOF > ./99-openshift-worker-tpmv2-encryption.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: worker-tpm
  labels:
    machineconfiguration.openshift.io/role: worker
spec:
  config:
```

```
    ignition:
      version: 2.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;base64,e30K
        filesystem: root
        mode: 420
        path: /etc/clevis.json
EOF
```

- To create a master file, run the following command:

```
$ cat << EOF > ./99-openshift-master-tpmv2-encryption.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: master-tpm
  labels:
    machineconfiguration.openshift.io/role: master
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;base64,e30K
        filesystem: root
        mode: 420
        path: /etc/clevis.json
EOF
```

4. Make a backup copy of the YAML file. You should do this because the file will be deleted when you create the cluster.

5. Continue with the remainder of the OpenShift Container Platform deployment.

3.2.3.2. Enabling Tang disk encryption

Use this procedure to enable Tang mode disk encryption during OpenShift Container Platform deployment.

Procedure

1. Access a Red Hat Enterprise Linux server from which you can configure the encryption settings and run **openshift-install** to install a cluster and **oc** to work with it.

2. Set up or access an existing Tang server. See Network-bound disk encryption for instructions. See Securing Automated Decryption New Cryptography and Techniques for a presentation on Tang.

3. Add kernel arguments to configure networking when you do the Red Hat Enterprise Linux CoreOS (RHCOS) installations for your cluster. For example, to configure DHCP networking, identify **ip=dhcp**, or set static networking when you add parameters to the kernel command line.

For both DHCP and static networking, you also must provide the **rd.neednet=1** kernel argument.

IMPORTANT

Skipping this step causes the second boot to fail.

4. Install the clevis package, if it is not already installed:

```
$ sudo yum install clevis -y
```

5. Generate a thumbprint from the Tang server.

   a. In the following command, replace the value of **url** with the Tang server URL:

   ```
   $ echo nifty random wordwords \
       | clevis-encrypt-tang \
         '{"url":"https://tang.example.org"}'
   ```

   Example output

   ```
   The advertisement contains the following signing keys:

   PLjNyRdGw03zlRoGjQYMahSZGu9
   ```

   b. When the **Do you wish to trust these keys? [ynYN]** prompt displays, type **Y**, and the thumbprint is displayed:

   Example output

   ```
   eyJhbmc3SlRyMXpPenc3ajhEQ01tZVJiTi1oM...
   ```

6. Create a Base64 encoded file, replacing the URL of the Tang server (**url**) and thumbprint (**thp**) you just generated:

   ```
   $ (cat <<EOM
   {
    "url": "https://tang.example.com",
    "thp": "PLjNyRdGw03zlRoGjQYMahSZGu9"
   }
   EOM
   ) | base64 -w0
   ```

   Example output

   ```
   ewogInVybCI6ICJodHRwczovL3RhbmcuZXhhbXBsZS5jb20iLAogInRocCI6ICJaUk1leTFjR3cw
   N3psVExHYlhuUWFoUzBHdTAiCn0K
   ```

7. In the **openshift** directory, create master or worker files to encrypt disks for those nodes.

   - For worker nodes, use the following command:

```
$ cat << EOF > ./99-openshift-worker-tang-encryption.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: worker-tang
  labels:
    machineconfiguration.openshift.io/role: worker
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;base64,e30K
          source:
data:text/plain;base64,ewogInVybCI6ICJodHRwczovL3RhbmcuZXhhbXBsZS5jb20iLAogIn
RocCI6ICJaUk1leTFjR3cwN3psVExHYlhuUWFoUzBHdTAiCn0K
        filesystem: root
        mode: 420
        path: /etc/clevis.json
EOF
```

- For master nodes, use the following command:

```
$ cat << EOF > ./99-openshift-master-tang-encryption.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: master-tang
  labels:
    machineconfiguration.openshift.io/role: master
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;base64,e30K
          source:
data:text/plain;base64,ewogInVybCI6ICJodHRwczovL3RhbmcuZXhhbXBsZS5jb20iLAogIn
RocCI6ICJaUk1leTFjR3cwN3psVExHYlhuUWFoUzBHdTAiCn0K
        filesystem: root
        mode: 420
        path: /etc/clevis.json
EOF
```

8. Add the **rd.neednet=1** kernel argument, as shown in the following example:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: <node_type>-tang <.>
spec:
```

```
  config:
    ignition:
      version: 3.1.0
  kernelArguments:
    - rd.neednet=1 <.>
```

Use the name you defined in the previous examples based on the type of node you are configuring, for example: **name: worker-tang**.

Required.

9. Continue with the remainder of the OpenShift Container Platform deployment.

3.2.4. Configuring chrony time service

You can set the time server and related settings used by the chrony time service (chronyd) by modifying the contents of the **chrony.conf** file and passing those contents to your nodes as a MachineConfig.

Procedure

1. Create the contents of the **chrony.conf** file and encode it as base64. For example:

```
$ cat << EOF | base64
   pool 0.rhel.pool.ntp.org iburst (1)
   driftfile /var/lib/chrony/drift
   makestep 1.0 3
   rtcsync
   logdir /var/log/chrony
   EOF
```

(1) Specify any valid, reachable time source. Alternately, you can specify any of the following NTP servers: **1.rhel.pool.ntp.org**, **2.rhel.pool.ntp.org**, or **3.rhel.pool.ntp.org**.

Example output

```
ICAgIHNlcnZlciBjbG9jay5yZWRoYXQuY29tIGlidXJzdAogICAgZHJpZnRmaWxlIC92YXIvbGli
L2Nocm9ueS9kcmlmdAogICAgbWFrZXN0ZXAgMS4wIDMKICAgIHJ0Y3N5bmMKICAgIGxvZ2Rp
RpciAv
dmFyL2xvZy9jaHJvbnkK
```

2. Create the MachineConfig file, replacing the base64 string with the one you just created yourself. This example adds the file to **master** nodes. You can change it to **worker** or make an additional MachineConfig for the **worker** role:

```
$ cat << EOF > ./masters-chrony-configuration.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: masters-chrony-configuration
spec:
  config:
```

```
    ignition:
      config: {}
      security:
        tls: {}
      timeouts: {}
      version: 2.2.0
    networkd: {}
    passwd: {}
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,ICAgIHNlcnZlciBjbG9jay5yZWRoYXQuY29tIGlidXJzdAogICAgZHJpZnRmaWxlIC92Y
XIvbGliL2Nocm9ueS9kcmlmdAogICAgbWFrZXN0ZXAgMS4wIDMKICAgIHJ0Y3N5bmMKICAg
IGxvZ2RpciAvdmFyL2xvZy9jaHJvbnkK
          verification: {}
        filesystem: root
        mode: 420
        path: /etc/chrony.conf
  osImageURL: ""
EOF
```

3. Make a backup copy of the configuration file.

4. If the cluster is not up yet, generate manifest files, add this file to the **openshift** directory, then continue to create the cluster.

5. If the cluster is already running, apply the file as follows:

```
$ oc apply -f ./masters-chrony-configuration.yaml
```

3.2.5. Additional resources

See Support for FIPS cryptography for information on FIPS support.

3.3. CREATING A MIRROR REGISTRY FOR INSTALLATION IN A RESTRICTED NETWORK

Before you install a cluster on infrastructure that you provision in a restricted network, you must mirror the required container images into that environment. Installations on a restricted network are supported on only infrastructure that you provision, not infrastructure that the installer provisions. You can also use this procedure in unrestricted networks to ensure your clusters only use container images that have satisfied your organizational controls on external content.

IMPORTANT

You must have access to the internet to obtain the necessary container images. In this procedure, you place the mirror registry on a mirror host that has access to both your network and the internet. If you do not have access to a mirror host, use the disconnected procedure to copy images to a device you can move across network boundaries with.

3.3.1. About the mirror registry

You can mirror the images that are required for OpenShift Container Platform installation and subsequent product updates to a mirror registry. These actions use the same process. The release image, which contains the description of the content, and the images it references are all mirrored. In addition, the Operator catalog source image and the images that it references must be mirrored for each Operator that you use. After you mirror the content, you configure each cluster to retrieve this content from your mirror registry.

The mirror registry can be any container registry that supports the most recent container image API, which is referred to as **schema2**. All major cloud provider registries, as well as Red Hat Quay, Artifactory, and the open source Docker distribution registry have the necessary support. Using one of these registries ensures that OpenShift Container Platform can verify the integrity of each image in disconnected environments.

The mirror registry must be reachable by every machine in the clusters that you provision. If the registry is unreachable installation, updating, or normal operations such as workload relocation might fail. For that reason, you must run mirror registries in a highly available way, and the mirror registries must at least match the production availability of your OpenShift Container Platform clusters.

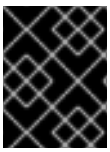
When you populate a mirror registry with OpenShift Container Platform images, you can follow two scenarios. If you have a host that can access both the internet and your mirror registry, but not your cluster nodes, you can directly mirror the content from that machine. This process is referred to as *connected mirroring*. If you have no such host, you must mirror the images to a file system and then bring that host or removable media into your restricted environment. This process is referred to as *disconnected mirroring*.

3.3.2. Preparing your mirror host

Before you perform the mirror procedure, you must prepare the host to retrieve content and push it to the remote location.

3.3.2.1. Installing the CLI by downloading the binary

You can install the OpenShift CLI (**oc**) in order to interact with OpenShift Container Platform from a command-line interface. You can install **oc** on Linux, Windows, or macOS.



IMPORTANT

If you installed an earlier version of **oc**, you cannot use it to complete all of the commands in OpenShift Container Platform 4.5. Download and install the new version of **oc**.

3.3.2.1.1. Installing the CLI on Linux

You can install the OpenShift CLI (**oc**) binary on Linux by using the following procedure.

Procedure

1. Navigate to the Infrastructure Provider page on the Red Hat OpenShift Cluster Manager site.
2. Select your infrastructure provider, and, if applicable, your installation type.
3. In the **Command-line interface** section, select **Linux** from the drop-down menu and click **Download command-line tools**.
4. Unpack the archive:

```
$ tar xvzf <file>
```

5. Place the **oc** binary in a directory that is on your **PATH**.
   To check your **PATH**, execute the following command:

```
$ echo $PATH
```

After you install the CLI, it is available using the **oc** command:

```
$ oc <command>
```

3.3.2.1.2. Installing the CLI on Windows

You can install the OpenShift CLI (**oc**) binary on Windows by using the following procedure.

Procedure

1. Navigate to the Infrastructure Provider page on the Red Hat OpenShift Cluster Manager site.
2. Select your infrastructure provider, and, if applicable, your installation type.
3. In the **Command-line interface** section, select **Windows** from the drop-down menu and click **Download command-line tools**.
4. Unzip the archive with a ZIP program.
5. Move the **oc** binary to a directory that is on your **PATH**.
   To check your **PATH**, open the command prompt and execute the following command:

```
C:\> path
```

After you install the CLI, it is available using the **oc** command:

```
C:\> oc <command>
```

3.3.2.1.3. Installing the CLI on macOS

You can install the OpenShift CLI (**oc**) binary on macOS by using the following procedure.

Procedure

1. Navigate to the Infrastructure Provider page on the Red Hat OpenShift Cluster Manager site.
2. Select your infrastructure provider, and, if applicable, your installation type.
3. In the **Command-line interface** section, select **MacOS** from the drop-down menu and click **Download command-line tools**.
4. Unpack and unzip the archive.
5. Move the **oc** binary to a directory on your PATH.
   To check your **PATH**, open a terminal and execute the following command:

```
$ echo $PATH
```

After you install the CLI, it is available using the **oc** command:

```
$ oc <command>
```

3.3.3. Configuring credentials that allow images to be mirrored

Create a container image registry credentials file that allows mirroring images from Red Hat to your mirror.

WARNING

Do not use this image registry credentials file as the pull secret when you install a cluster. If you provide this file when you install cluster, all of the machines in the cluster will have write access to your mirror registry.

WARNING

This process requires that you have write access to a container image registry on the mirror registry and adds the credentials to a registry pull secret.

Prerequisites

- You configured a mirror registry to use in your restricted network.
- You identified an image repository location on your mirror registry to mirror images into.
- You provisioned a mirror registry account that allows images to be uploaded to that image repository.

Procedure

Complete the following steps on the installation host:

1. Download your **registry.redhat.io** pull secret from the Pull Secret page on the Red Hat OpenShift Cluster Manager site and save it to a **.json** file.
2. Generate the base64-encoded user name and password or token for your mirror registry:

```
$ echo -n '<user_name>:<password>' | base64 -w0 (1)
BGVtbYk3ZHAtqXs=
```

(1) For **<user_name>** and **<password>**, specify the user name and password that you configured for your registry.

3. Make a copy of your pull secret in JSON format:

```
$ cat ./pull-secret.text | jq .  > <path>/<pull-secret-file>
```

(1) Specify the path to the folder to store the pull secret in and a name for the JSON file that you create.

The contents of the file resemble the following example:

```
{
  "auths": {
    "cloud.openshift.com": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "quay.io": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "registry.connect.redhat.com": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    },
    "registry.redhat.io": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    }
  }
}
```

4. Edit the new file and add a section that describes your registry to it:

```
  "auths": {
    "<mirror_registry>": { (1)
      "auth": "<credentials>", (2)
      "email": "you@example.com"
  },
```

(1) For **<mirror_registry>**, specify the registry domain name, and optionally the port, that your mirror registry uses to serve content. For example, **registry.example.com** or **registry.example.com:5000**

(2) For **<credentials>**, specify the base64-encoded user name and password for the mirror registry.

The file resembles the following example:

```
{
  "auths": {
    "<mirror_registry>": {
      "auth": "<credentials>",
      "email": "you@example.com"
    },
```

```
    "cloud.openshift.com": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "quay.io": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "registry.connect.redhat.com": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    },
    "registry.redhat.io": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    }
  }
}
```

3.3.4. Mirroring the OpenShift Container Platform image repository

Mirror the OpenShift Container Platform image repository to your registry to use during cluster installation or upgrade.

Prerequisites

- Your mirror host has access to the Internet.
- You configured a mirror registry to use in your restricted network and can access the certificate and credentials that you configured.
- You downloaded the pull secret from the Pull Secret page on the Red Hat OpenShift Cluster Manager site and modified it to include authentication to your mirror repository.

Procedure

Complete the following steps on the mirror host:

1. Review the OpenShift Container Platform downloads page to determine the version of OpenShift Container Platform that you want to install and determine the corresponding tag on the Repository Tags page.
2. Set the required environment variables:
   a. Export the release version:

      ```
      $ OCP_RELEASE=<release_version>
      ```

      For **<release_version>**, specify the tag that corresponds to the version of OpenShift Container Platform to install, such as **4.5.4**.

   b. Export the local registry name and host port:

      ```
      $ LOCAL_REGISTRY='<local_registry_host_name>:<local_registry_host_port>'
      ```

For **<local_registry_host_name>**, specify the registry domain name for your mirror repository, and for **<local_registry_host_port>**, specify the port that it serves content on.

c. Export the local repository name:

```
$ LOCAL_REPOSITORY='<local_repository_name>'
```

For **<local_repository_name>**, specify the name of the repository to create in your registry, such as **ocp4/openshift4**.

d. Export the name of the repository to mirror:

```
$ PRODUCT_REPO='openshift-release-dev'
```

For a production release, you must specify **openshift-release-dev**.

e. Export the path to your registry pull secret:

```
$ LOCAL_SECRET_JSON='<path_to_pull_secret>'
```

For **<path_to_pull_secret>**, specify the absolute path to and file name of the pull secret for your mirror registry that you created.

f. Export the release mirror:

```
$ RELEASE_NAME="ocp-release"
```

For a production release, you must specify **ocp-release**.

g. Export the type of architecture for your server, such as **x86_64**.:

```
$ ARCHITECTURE=<server_architecture>
```

h. Export the path to the directory to host the mirrored images:

```
$ REMOVABLE_MEDIA_PATH=<path> 1
```

1 Specify the full path, including the initial forward slash (/) character.

3. Mirror the version images to the internal container registry:

- If your mirror host does not have Internet access, take the following actions:

  i. Connect the removable media to a system that is connected to the Internet.

  ii. Review the images and configuration manifests to mirror:

```
$ oc adm -a ${LOCAL_SECRET_JSON} release mirror \
    --from=quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE}-
${ARCHITECTURE} \
    --to=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} \
    --to-release-
image=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
${ARCHITECTURE} --dry-run
```

iii. Record the entire **imageContentSources** section from the output of the previous command. The information about your mirrors is unique to your mirrored repository, and you must add the **imageContentSources** section to the **install-config.yaml** file during installation.

iv. Mirror the images to a directory on the removable media:

```
$ oc adm release mirror -a ${LOCAL_SECRET_JSON} --to-
dir=${REMOVABLE_MEDIA_PATH}/mirror
quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE}-
${ARCHITECTURE}
```

v. Take the media to the restricted network environment and upload the images to the local container registry.

```
$ oc image mirror -a ${LOCAL_SECRET_JSON} --from-
dir=${REMOVABLE_MEDIA_PATH}/mirror
"file://openshift/release:${OCP_RELEASE}*"
${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} 1
```

1 For **REMOVABLE_MEDIA_PATH**, you must use the same path that you specified when you mirrored the images.

- If the local container registry is connected to the mirror host, take the following actions:

i. Directly push the release images to the local registry by using following command:

```
$ oc adm -a ${LOCAL_SECRET_JSON} release mirror \
    --from=quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE}-
${ARCHITECTURE} \
    --to=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} \
    --to-release-
image=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
${ARCHITECTURE}
```

This command pulls the release information as a digest, and its output includes the **imageContentSources** data that you require when you install your cluster.

ii. Record the entire **imageContentSources** section from the output of the previous command. The information about your mirrors is unique to your mirrored repository, and you must add the **imageContentSources** section to the **install-config.yaml** file during installation.

NOTE

The image name gets patched to Quay.io during the mirroring process, and the podman images will show Quay.io in the registry on the bootstrap virtual machine.

4. To create the installation program that is based on the content that you mirrored, extract it and pin it to the release:

- If your mirror host does not have Internet access, run the following command:

```
$ oc adm -a ${LOCAL_SECRET_JSON} release extract --command=openshift-install
"${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}"
```

- If the local container registry is connected to the mirror host, run the following command:

```
$ oc adm -a ${LOCAL_SECRET_JSON} release extract --command=openshift-install
"${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
${ARCHITECTURE}"
```

IMPORTANT

To ensure that you use the correct images for the version of OpenShift Container Platform that you selected, you must extract the installation program from the mirrored content.

You must perform this step on a machine with an active Internet connection.

3.3.5. Preparing your cluster to gather support data

Clusters using a restricted network must import the default must-gather image in order to gather debugging data for Red Hat support. The must-gather image is not imported by default, and clusters on a restricted network do not have access to the internet to pull the latest image from a remote repository.

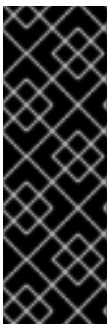
Procedure

1. Import the default must-gather image from your installation payload:

```
$ oc import-image is/must-gather -n openshift
```

3.3.6. Using Cluster Samples Operator imagestreams with alternate or mirrored registries

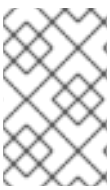
Most imagestreams in the **openshift** namespace managed by the Cluster Samples Operator point to images located in the Red Hat registry at registry.redhat.io. Mirroring will not apply to these imagestreams.



IMPORTANT

The **jenkins**, **jenkins-agent-maven**, and **jenkins-agent-nodejs** imagestreams come from the install payload and are managed by the Samples Operator, so no further mirroring procedures are needed for those imagestreams.

Setting the **samplesRegistry** field in the Sample Operator configuration file to registry.redhat.io is redundant because it is already directed to registry.redhat.io for everything but Jenkins images and imagestreams.



NOTE

The **cli**, **installer**, **must-gather**, and **tests** imagestreams, while part of the install payload, are not managed by the Cluster Samples Operator. These are not addressed in this procedure.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- Create a pull secret for your mirror registry.

Procedure

1. Access the images of a specific imagestream to mirror, for example:

```
$ oc get is <imagestream> -n openshift -o json | jq .spec.tags[].from.name | grep
registry.redhat.io
```

2. Mirror images from registry.redhat.io associated with any imagestreams you need in the restricted network environment into one of the defined mirrors, for example:

```
$ oc image mirror registry.redhat.io/rhscl/ruby-25-rhel7:latest ${MIRROR_ADDR}/rhscl/ruby-
25-rhel7:latest
```

3. Create the cluster's image configuration object:

```
$ oc create configmap registry-config --from-
file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
```

4. Add the required trusted CAs for the mirror in the cluster's image configuration object:

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
{"name":"registry-config"}}}' --type=merge
```

5. Update the **samplesRegistry** field in the Cluster Samples Operator configuration object to contain the **hostname** portion of the mirror location defined in the mirror configuration:

```
$ oc edit configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```

NOTE

This is required because the imagestream import process does not use the mirror or search mechanism at this time.

6. Add any imagestreams that are not mirrored into the **skippedImagestreams** field of the Cluster Samples Operator configuration object. Or if you do not want to support any of the sample imagestreams, set the Cluster Samples Operator to **Removed** in the Cluster Samples Operator configuration object.

NOTE

Any unmirrored imagestreams that are not skipped, or if the Samples Operator is not changed to **Removed**, will result in the Samples Operator reporting a **Degraded** status two hours after the imagestream imports start failing.

Many of the templates in the **openshift** namespace reference the imagestreams. So using **Removed** to purge both the imagestreams and templates will eliminate the possibility of attempts to use them if they are not functional because of any missing imagestreams.

3.3.7. Next steps

- Install a cluster on infrastructure that you provision in your restricted nework, such as on VMware vSphere, bare metal, or Amazon Web Services.

3.4. AVAILABLE CLUSTER CUSTOMIZATIONS

You complete most of the cluster configuration and customization after you deploy your OpenShift Container Platform cluster. A number of *configuration resources* are available.

You modify the configuration resources to configure the major features of the cluster, such as the image registry, networking configuration, image build behavior, and the identity provider.

For current documentation of the settings that you control by using these resources, use the **oc explain** command, for example **oc explain builds --api-version=config.openshift.io/v1**

3.4.1. Cluster configuration resources

All cluster configuration resources are globally scoped (not namespaced) and named **cluster**.

Resource name	Description
apiserver.config.openshift.io	Provides api-server configuration such as certificates and certificate authorities.
authentication.config.openshift.io	Controls the identity providerand authentication configuration for the cluster.
build.config.openshift.io	Controls default and enforced configuration for all builds on the cluster.
console.config.openshift.io	Configures the behavior of the web console interface, including the logout behavior.
featuregate.config.openshift.io	Enables FeatureGates so that you can use Tech Preview features.
image.config.openshift.io	Configures how specific image registries should be treated (allowed, disallowed, insecure, CA details).
ingress.config.openshift.io	Configuration details related to routing such as the default domain for routes.
oauth.config.openshift.io	Configures identity providers and other behavior related to internal OAuth server flows.

Resource name	Description
project.config.openshift.io	Configures how projects are created including the project template.
proxy.config.openshift.io	Defines proxies to be used by components needing external network access. Note: not all components currently consume this value.
scheduler.config.openshift.io	Configures scheduler behavior such as policies and default nodeselectors.

3.4.2. Operator configuration resources

These configuration resources are cluster-scoped instances, named **cluster**, which control the behavior of a specific component as owned by a particular operator.

Resource name	Description
console.operator.openshift.io	Controls console appearance such as branding customizations
config.imageregistry.operator.openshift.io	Configures internal image registry settings such as public routing, log levels, proxy settings, resource constraints, replica counts, and storage type.
config.samples.operator.openshift.io	Configures the Samples Operator to control which example imagestreams and templates are installed on the cluster.

3.4.3. Additional configuration resources

These configuration resources represent a single instance of a particular component. In some cases, you can request multiple instances by creating multiple instances of the resource. In other cases, the Operator can use only a specific resource instance name in a specific namespace. Reference the component-specific documentation for details on how and when you can create additional resource instances.

Resource name	Instance name	Namespace	Description
alertmanager.monitoring.coreos.com	main	openshift-monitoring	Controls the alertmanager deployment parameters.
ingresscontroller.operator.openshift.io	default	openshift-ingress-operator	Configures Ingress Operator behavior such as domain, number of replicas, certificates, and controller placement.

3.4.4. Informational Resources

You use these resources to retrieve information about the cluster. Do not edit these resources directly.

Resource name	Instance name	Description
clusterversion.config.openshift.io	version	In OpenShift Container Platform 4.5, you must not customize the ClusterVersion resource for production clusters. Instead, follow the process to update a cluster.
dns.config.openshift.io	cluster	You cannot modify the DNS settings for your cluster. You can view the DNS Operator status.
infrastructure.config.openshift.io	cluster	Configuration details allowing the cluster to interact with its cloud provider.
network.config.openshift.io	cluster	You cannot modify your cluster networking after installation. To customize your network, follow the process to customize networking during installation.

3.4.5. Updating the global cluster pull secret

You can update the global pull secret for your cluster.

WARNING

Cluster resources must adjust to the new pull secret, which can temporarily limit the usability of the cluster.

WARNING

Updating the global pull secret will cause node reboots while the Machine Config Operator (MCO) syncs the changes.

Prerequisites

- You have a new or modified pull secret file to upload.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

- Enter the following command to update the global pull secret for your cluster:

```
$ oc set data secret/pull-secret -n openshift-config --from-file=.dockerconfigjson=<pull-secret-location> 1
```

1 Provide the path to the new pull secret file.

This update is rolled out to all nodes, which can take some time depending on the size of your cluster. During this time, nodes are drained and pods are rescheduled on the remaining nodes.

3.5. CONFIGURING YOUR FIREWALL

If you use a firewall, you must configure it so that OpenShift Container Platform can access the sites that it requires to function. You must always grant access to some sites, and you grant access to more if you use Red Hat Insights, the Telemetry service, a cloud to host your cluster, and certain build strategies.

3.5.1. Configuring your firewall for OpenShift Container Platform

Before you install OpenShift Container Platform, you must configure your firewall to grant access to the sites that OpenShift Container Platform requires.

There are no special configuration considerations for services running on only controller nodes versus worker nodes.

Procedure

1. Allowlist the following registry URLs:

URL	Function
registry.redhat.io	Provides core container images
quay.io	Provides core container images
sso.redhat.com	The **https://cloud.redhat.com/openshift** site uses authentication from **sso.redhat.com**
openshift.org	Provides Red Hat Enterprise Linux CoreOS (RHCOS) images

2. Allowlist any site that provides resources for a language or framework that your builds require.

3. If you do not disable Telemetry, you must grant access to the following URLs to access Red Hat Insights:

URL	Function
cert-api.access.redhat.com	Required for Telemetry
api.access.redhat.com	Required for Telemetry

URL	Function
infogw.api.openshift.com	Required for Telemetry
https://cloud.redhat.com/api/ingress	Required for Telemetry and for **insights-operator**

4. If you use Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform (GCP) to host your cluster, you must grant access to the URLs that provide the cloud provider API and DNS for that cloud:

Cloud	URL	Function
AWS	***.amazonaws.com**	Required to access AWS services and resources. Review the AWS Service Endpoints in the AWS documentation to determine the exact endpoints to allow for the regions that you use.
	oso-rhc4tp-docker-registry.s3-us-west-2.amazonaws.com	Required to access AWS services and resources when using strict security requirements. Review the AWS Service Endpoints in the AWS documentation to determine the exact endpoints to allow for the regions that you use.
GCP	***.googleapis.com**	Required to access GCP services and resources. Review Cloud Endpoints in the GCP documentation to determine the endpoints to allow for your APIs.
	accounts.google.com	Required to access your GCP account.
Azure	**management.azure.com**	Required to access Azure services and resources. Review the Azure REST API Reference in the Azure documentation to determine the endpoints to allow for your APIs.

5. Allowlist the following URLs:

URL	Function
mirror.openshift.com	Required to access mirrored installation content and images. This site is also a source of release image signatures, although the Cluster Version Operator needs only a single functioning source.

URL	Function
storage.googleapis.com/openshift-release	A source of release image signatures, although the Cluster Version Operator needs only a single functioning source.
***.apps.<cluster_name>.<base_domain>**	Required to access the default cluster routes unless you set an ingress wildcard during installation.
quay-registry.s3.amazonaws.com	Required to access Quay image content in AWS.
api.openshift.com	Required to check if updates are available for the cluster.
art-rhcos-ci.s3.amazonaws.com	Required to download Red Hat Enterprise Linux CoreOS (RHCOS) images.
api.openshift.com	Required for your cluster token.
cloud.redhat.com/openshift	Required for your cluster token.

Operators require route access to perform health checks. Specifically, the authentication and web console Operators connect to two routes to verify that the routes work. If you are the cluster administrator and do not want to allow ***.apps.<cluster_name>.<base_domain>**, then allow these routes:

- **oauth-openshift.apps.<cluster_name>.<base_domain>**
- **console-openshift-console.apps.<cluster_name>.<base_domain>**, or the host name that is specified in the **spec.route.hostname** field of the **consoles.operator/cluster** object if the field is not empty.

3.6. CONFIGURING A PRIVATE CLUSTER

After you install an OpenShift Container Platform version 4.5 cluster, you can set some of its core components to be private.

IMPORTANT

You can configure this change for only clusters that use infrastructure that you provision to a cloud provider.

3.6.1. About private clusters

By default, OpenShift Container Platform is provisioned using publicly-accessible DNS and endpoints. You can set the DNS, Ingress Controller, and API server to private after you deploy your cluster.

DNS

If you install OpenShift Container Platform on installer-provisioned infrastructure, the installation program creates records in a pre-existing public zone and, where possible, creates a private zone for the cluster's own DNS resolution. In both the public zone and the private zone, the installation program or cluster creates DNS entries for ***.apps**, for Ingress, and **api**, for the API server.

The ***.apps** records in the public and private zone are identical, so when you delete the public zone, the private zone seamlessly provides all DNS resolution for the cluster.

Ingress Controller
Because the default Ingress object is created as public, the load balancer is internet-facing and in the public subnets. You can replace the default Ingress Controller with an internal one.

API server
By default, the installation program creates appropriate network load balancers for the API server to use for both internal and external traffic.

On Amazon Web Services (AWS), separate public and private load balancers are created. The load balancers are identical except that an additional port is available on the internal one for use within the cluster. Although the installation program automatically creates or destroys the load balancer based on API server requirements, the cluster does not manage or maintain them. As long as you preserve the cluster's access to the API server, you can manually modify or move the load balancers. For the public load balancer, port 6443 is open and the health check is configured for HTTPS against the /**readyz** path.

On Google Cloud Platform, a single load balancer is created to manage both internal and external API traffic, so you do not need to modify the load balancer.

On Microsoft Azure, both public and private load balancers are created. However, because of limitations in current implementation, you just retain both load balancers in a private cluster.

3.6.2. Setting DNS to private

After you deploy a cluster, you can modify its DNS to use only a private zone.

Procedure

1. Review the DNS custom resource for your cluster:

```
$ oc get dnses.config.openshift.io/cluster -o yaml
```

Example output

```
apiVersion: config.openshift.io/v1
kind: DNS
metadata:
  creationTimestamp: "2019-10-25T18:27:09Z"
  generation: 2
  name: cluster
  resourceVersion: "37966"
  selfLink: /apis/config.openshift.io/v1/dnses/cluster
  uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
spec:
  baseDomain: <base_domain>
  privateZone:
    tags:
      Name: <infrastructureID>-int
```

```
    kubernetes.io/cluster/<infrastructureID>: owned
  publicZone:
    id: Z2XXXXXXXXXXA4
status: {}
```

Note that the **spec** section contains both a private and a public zone.

2. Patch the DNS custom resource to remove the public zone:

```
$ oc patch dnses.config.openshift.io/cluster --type=merge --patch='{"spec": {"publicZone":
null}}'
dns.config.openshift.io/cluster patched
```

Because the Ingress Controller consults the DNS definition when it creates Ingress objects, When you create or modify Ingress objects, only private records are created.

IMPORTANT

DNS records for the existing Ingress objects are not modified when you remove the public zone.

3. Optional: Review the DNS custom resource for your cluster and confirm that the public zone was removed:

```
$ oc get dnses.config.openshift.io/cluster -o yaml
```

Example output

```
apiVersion: config.openshift.io/v1
kind: DNS
metadata:
  creationTimestamp: "2019-10-25T18:27:09Z"
  generation: 2
  name: cluster
  resourceVersion: "37966"
  selfLink: /apis/config.openshift.io/v1/dnses/cluster
  uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
spec:
  baseDomain: <base_domain>
  privateZone:
    tags:
      Name: <infrastructureID>-int
      kubernetes.io/cluster/<infrastructureID>-wfpg4: owned
status: {}
```

3.6.3. Setting the Ingress Controller to private

After you deploy a cluster, you can modify its Ingress Controller to use only a private zone.

Procedure

1. Modify the default Ingress Controller to use only an internal endpoint:

```
$ oc replace --force --wait --filename - <<EOF
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: default
spec:
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal
EOF
```

Example output

```
ingresscontroller.operator.openshift.io "default" deleted
ingresscontroller.operator.openshift.io/default replaced
```

The public DNS entry is removed, and the private zone entry is updated.

3.6.4. Restricting the API server to private

After you deploy a cluster to Amazon Web Services (AWS) or Microsoft Azure, you can reconfigure the API server to use only the private zone.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Have access to the web console as a user with **admin** privileges.

Procedure

1. In the web portal or console for AWS or Azure, take the following actions:
   a. Locate and delete appropriate load balancer component.
      - For AWS, delete the external load balancer. The API DNS entry in the private zone already points to the internal load balancer, which uses an identical configuration, so you do not need to modify the internal load balancer.
      - For Azure, delete the **api-internal** rule for the load balancer.
   b. Delete the **api.$clustername.$yourdomain** DNS entry in the public zone.
2. From your terminal, list the cluster machines:

```
$ oc get machine -n openshift-machine-api
```

Example output

```
NAME                 STATE    TYPE       REGION     ZONE         AGE
lk4pj-master-0         running  m4.xlarge  us-east-1  us-east-1a   17m
lk4pj-master-1         running  m4.xlarge  us-east-1  us-east-1b   17m
```

```
lk4pj-master-2                running   m4.xlarge   us-east-1   us-east-1a   17m
lk4pj-worker-us-east-1a-5fzfj   running   m4.xlarge   us-east-1   us-east-1a   15m
lk4pj-worker-us-east-1a-vbghs   running   m4.xlarge   us-east-1   us-east-1a   15m
lk4pj-worker-us-east-1b-zgpzg   running   m4.xlarge   us-east-1   us-east-1b   15m
```

You modify the control plane machines, which contain **master** in the name, in the following step.

3. Remove the external load balancer from each control plane machine.

   a. Edit a **master** Machine object to remove the reference to the external load balancer.

   ```
   $ oc edit machines -n openshift-machine-api <master_name> 1
   ```

   1 Specify the name of the control plane, or master, Machine to modify.

   b. Remove the lines that describe the external load balancer, which are marked in the following example, and save and exit the object specification:

   ```
   ...
   spec:
     providerSpec:
       value:
       ...
         loadBalancers:
         - name: lk4pj-ext 1
           type: network 2
         - name: lk4pj-int
           type: network
   ```

   1 2 Delete this line.

   c. Repeat this process for each of the machines that contains **master** in the name.