OpenShift Container Platform 4.3

Installing

Installing and configuring OpenShift Container Platform clusters

Last Updated: 2020-03-18

OpenShift Container Platform 4.3 Installing

Installing and configuring OpenShift Container Platform clusters

Legal Notice

Abstract

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

Table of Contents

CHAPTER 1. GATHERING INSTALLATION LOGS

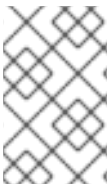
To assist in troubleshooting a failed OpenShift Container Platform installation, you can gather logs from the bootstrap and control plane, or master, machines.

Prerequisites

- You attempted to install an OpenShift Container Platform cluster, and installation failed.
- You provided an SSH key to the installation program, and that key is in your running **ssh-agent** process.

1.1. 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 must be running and accessible through SSH.
- The **ssh-agent** process is active on your computer, and you provided both the **ssh-agent** process and the installation program the same SSH key.
- If you tried to install a cluster on infrastructure that you provisioned, you must have the fully-qualified domain names of the control plane, or master, machines.

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=<directory> 1
     ```

     1 **installation_directory** is the directory you stored 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=<directory> \ 1
       --bootstrap <bootstrap_address> \ 2
       --master <master_1_address> \ 3
     ```

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

1. **installation_directory** is the directory you stored 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.

The command output resembles the following example:

```
INFO Pulling debug logs from the bootstrap machine
INFO Bootstrap gather logs captured here "<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.

1.2. 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/*
```

1.3. 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
```

CHAPTER 2. SUPPORT FOR FIPS CRYPTOGRAPHY

Starting with version 4.3, you can install an OpenShift Container Platform cluster that uses FIPS validated / Implementation Under Test 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 / Implementation Under Test 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 / Implementation Under Test 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, Red Hat's golang compiler calls RHEL and RHCOS cryptographic libraries for all OpenShift Container Platform components that require cryptographic signing. At the initial release of OpenShift Container Platform version 4.3, only the **ose-sdn** package uses the native golang cryptography, which is not FIPS validated / Implementation Under Test. Red Hat verifies that all other packages use the FIPS validated / Implementation Under Test OpenSSL module.

Table 2.1. FIPS mode attributes and limitations in OpenShift Container Platform 4.3

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 / Implementation Under Test 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 / Implementation Under Test modules, ensure that the systems that support your OpenShift Container Platform cluster use FIPS validated / Implementation Under Test modules for cryptography.

2.2.1. etcd

To ensure that the secrets that are stored in etcd use FIPS validated / Implementation Under Test encryption, encrypt the etcd datastore by using a FIPS-approved cryptographic algorithm. After you install the cluster, you can encrypt the etcd data by using the **aes cbc** 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 / Implementation Under Test 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 has is using FIPS validated / Implementation Under Test 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.

Table 3.1. Installer-provisioned infrastructure options

	AWS	Azure	GCP	OpenStack	Bare metal	vSphere	IBM Z
Default	X	X	X				
Custom	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	Bare metal	vSphere	IBM Z
Custom	X		X		X	X	
Network Operator					X	X	
Restricted network	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. Get a RHEL 8 system, then register and subscribe it:

```
# subscription-manager register
Username: yourname
Password: ***************
# subscription-manager attach --auto
```

2. Install software needed to build the software and container:

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

3. Clone the kmod-via-containers repository:

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

4. 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:

```
$ cd kmods-via-containers/
$ sudo make install
$ sudo systemctl daemon-reload
```

5. 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
```

6. Edit the configuration file, **simple-kmod.conf**, in his example, and change the name of the Dockerfile to **Dockerfile.rhel** so the file appears as shown here:

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

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"
```

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

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

8. Enable and start the systemd service, then check the status:

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

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

```
$ lsmod | grep simple_
simple_procfs_kmod     16384  0
simple_kmod            16384  0
```

10. The simple-kmod example has a few other ways to test that it is working. Look for a "Hello world" message in the kernel ring buffer with **dmesg**:

```
$ dmesg | grep 'Hello world'
[ 6420.761332] Hello world from simple_kmod.
```

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

```
$ sudo cat /proc/simple-procfs-kmod
simple-procfs-kmod number = 0
```

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

```
$ sudo spkut 44
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.

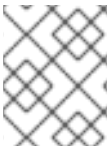
- Provide RHEL entitlements to each node.
- Get RHEL entitlements from an existing RHEL host, from the **/etc/pki/entitlement** directory and copy them to the same location as the other files you provide when you build your Ignition config.
- Inside the Dockerfile, add pointers to a **yum** repository 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. Get a RHEL 8 system, then register and subscribe it:

```
# subscription-manager register
Username: yourname
Password: ***************
# subscription-manager attach --auto
```

2. Install software needed to build the software:

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

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

```
$ mkdir kmods; cd kmods
$ 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=multi-
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.

4. Create a base MCO YAML snippet that looks like:

```
$ 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.

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

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

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

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

```
$ FAKEROOT=$(mktemp -d)
$ cd kmods-via-containers
$ make install DESTDIR=${FAKEROOT}/usr/local CONFDIR=${FAKEROOT}/etc/
$ cd ../kvc-simple-kmod
$ make install DESTDIR=${FAKEROOT}/usr/local CONFDIR=${FAKEROOT}/etc/
```

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

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

5. 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
```

6. 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.

7. 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_
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 a master and/or worker file to encrypt disks for those nodes. Here are examples of those two files:

```
$ 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
```

```
$ 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 RHCOS installations for your cluster. If you miss this step, the second boot will fail. For example, to configure DHCP networking, identify **ip=dhcp** or set static networking when you add parameters to the kernel command line.

4. Generate the thumbprint. Install the clevis package, it is not already installed, and generate a thumbprint from the Tang server. Replace the value of **url** with the Tang server URL:

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

The advertisement contains the following signing keys:

PLjNyRdGw03zlRoGjQYMahSZGu9

Do you wish to trust these keys? [ynYN] y
eyJhbmc3SlRyMXpPenc3ajhEQ01tZVJiTi1oM...
```

5. 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

ewogInVybCI6ICJodHRwczovL3RhbmcuZXhhbXBsZS5jb20iLAogInRocCI6ICJaUk1leTFjR3cw
N3psVExHYlhuUWFoUzBHdTAiCn0K
```

6. Replace the “source” in the TPM2 example with the Base64 encoded file for one or both of these examples for worker and/or master nodes:

```
$ 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,ewogInVybCI6ICJodHRwczovL3RhbmcuZXhhbXBsZS5jb20iLAogInRoc
CI6ICJaUk1leTFjR3cwN3psVExxHYlhuUWFoUzBHdTAiCn0K
        filesystem: root
        mode: 420
        path: /etc/clevis.json
EOF
```

```
$ cat << EOF > ./99_openshift-master-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,ewogInVybCI6ICJodHRwczovL3RhbmcuZXhhbXBsZS5jb20iLAogInRoc
CI6ICJaUk1leTFjR3cwN3psVExxHYlhuUWFoUzBHdTAiCn0K
        filesystem: root
        mode: 420
        path: /etc/clevis.json
EOF
```

7. 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
    server clock.redhat.com iburst
    driftfile /var/lib/chrony/drift
    makestep 1.0 3
    rtcsync
    logdir /var/log/chrony
    EOF
```

```
ICAgIHNlcnZlciBjbG9jay5yZWRoYXQuY29tIGlidXJzdAogICAgZHJpZnRmaWxlIC92YXIvbGli
L2Nocm9ueS9kcmlmdAogICAgbWFrZXN0ZXAgMS4wIDMKICAgIHJ0Y3N5bmMKICAgIGxvZ2
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 > ./99_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,c2VydmVyIGNsb2NrLnJlZGhhdC5jb20gaWJ1cnN0CmRyaWZ0ZmlsZSAvdmFyL2xp
Yi9jaHJvbnkvZHJpZnQKbWFrZXN0ZXAgMS4wIDMKcnRjc3luYwpsb2dkaXIgL3Zhci9sb2cvY2h
yb255Cg==
          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 create a mirror registry. Installations on a restricted network are supported on only infrastructure that you provision, not infrastructure that the installer provisions.

IMPORTANT

You must have access to the internet to obtain the data that populates the mirror repository. In this procedure, you place the mirror registry on a bastion host that has access to both your network and the internet. If you do not have access to a bastion host, use the method that best fits your restrictions to bring the contents of the mirror registry into your restricted network.

3.3.1. About the mirror registry

You can mirror the contents of the OpenShift Container Platform registry and the images that are required to generate the installation program.

The mirror registry is a key component that is required to complete an installation in a restricted network. You can create this mirror on a bastion host, which can access both the internet and your closed network, or by using other methods that meet your restrictions.

Because of the way that OpenShift Container Platform verifies integrity for the release payload, the image references in your local registry are identical to the ones that are hosted by Red Hat on Quay.io. During the bootstrapping process of installation, the images must have the same digests no matter which repository they are pulled from. To ensure that the release payload is identical, you mirror the images to your local repository.

3.3.2. Preparing the bastion host

Before you create the mirror registry, you must prepare the bastion host.

3.3.2.1. Installing the CLI by downloading the binary

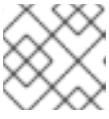
You can install the CLI in order to interact with OpenShift Container Platform using a command-line interface.

IMPORTANT

If you installed an earlier version of **oc**, you cannot use it to complete all of the commands in OpenShift Container Platform 4.3. Download and install the new version of **oc**.

Procedure

1. From the Infrastructure Provider page on the Red Hat OpenShift Cluster Manager site, navigate to the page for your installation type and click **Download Command-line Tools**.
2. Click the folder for your operating system and architecture and click the compressed file.



NOTE

You can install **oc** on Linux, Windows, or macOS.

3. Save the file to your file system.
4. Extract the compressed file.
5. Place it in a directory that is on your **PATH**.

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

```
$ oc <command>
```

3.3.3. Creating a mirror registry

Create a registry to host the mirrored content that you require for installing OpenShift Container Platform. For installation in a restricted network, you must place the mirror on your bastion host.

NOTE

The following procedure creates a simple registry that stores data in the **/opt/registry** folder and runs in a **podman** container. You can use a different registry solution, such as Red Hat Quay. Review the following procedure to ensure that your registry functions correctly.

Prerequisites

- You have a Red Hat Enterprise Linux (RHEL) server on your network to use as the registry host.
- The registry host can access the internet.

Procedure

On the bastion host, take the following actions:

1. Install the required packages:

   ```
   # yum -y install podman httpd-tools
   ```

   The **podman** package provides the container package that you run the registry in. The **httpd-tools** package provides the **htpasswd** utility, which you use to create users.

2. Create folders for the registry:

   ```
   # mkdir -p /opt/registry/{auth,certs,data}
   ```

   These folders are mounted inside the registry container.

3. Provide a certificate for the registry. If you do not have an existing, trusted certificate authority, you can generate a self-signed certificate:

   ```
   $ cd /opt/registry/certs
   # openssl req -newkey rsa:4096 -nodes -sha256 -keyout domain.key -x509 -days 365 -out
   domain.crt
   ```

   At the prompts, provide the required values for the certificate:

Country Name (2 letter code)	Specify the two-letter ISO country code for your location. See the ISO 3166 country codes standard.
State or Province Name (full name)	Enter the full name of your state or province.
Locality Name (eg, city)	Enter the name of your city.
Organization Name (eg, company)	Enter your company name.
Organizational Unit Name (eg, section)	Enter your department name.
Common Name (eg, your name or your server's hostname)	Enter the host name for the registry host. Ensure that your hostname is in DNS and that it resolves to the expected IP address.
Email Address	Enter your email address. For more information, see the req description in the OpenSSL documentation.

4. Generate a user name and a password for your registry that uses the **bcrpt** format:

```
# htpasswd -bBc /opt/registry/auth/htpasswd <user_name> <password> 1
```

1 Replace **<user_name>** and **<password>** with a user name and a password.

5. Create the **mirror-registry** container to host your registry:

```
# podman run --name mirror-registry -p <local_registry_host_port>:5000 \ 1
    -v /opt/registry/data:/var/lib/registry:z \
    -v /opt/registry/auth:/auth:z \
    -e "REGISTRY_AUTH=htpasswd" \
    -e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" \
    -e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd \
    -v /opt/registry/certs:/certs:z \
```

```
    -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt \
    -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key \
    -d docker.io/library/registry:2
```

1 For **<local_registry_host_port>**, specify the port that your mirror registry uses to serve content.

6. Open the required ports for your registry:

```
# firewall-cmd --add-port=<local_registry_host_port>/tcp --zone=internal --permanent (1)
# firewall-cmd --add-port=<local_registry_host_port>/tcp --zone=public   --permanent (2)
# firewall-cmd --reload
```

1 2 For **<local_registry_host_port>**, specify the port that your mirror registry uses to serve content.

7. Add the self-signed certificate to your list of trusted certificates:

```
# cp /opt/registry/certs/domain.crt /etc/pki/ca-trust/source/anchors/
# update-ca-trust
```

You must trust your certificate to log in to your registry during the mirror process.

8. Confirm that the registry is available:

```
$ curl -u <user_name>:<password> -k https://<local_registry_host_name>:
<local_registry_host_port>/v2/_catalog (1)

{"repositories":[]}
```

1 For **<user_name>** and **<password>**, specify the user name and password for your registry. For **<local_registry_host_name>**, specify the registry domain name that you specified in your certificate, such as **registry.example.com**. For **<local_registry_host_port>**, specify the port that your mirror registry uses to serve content.

If the command output displays an empty repository, your registry is available.

3.3.4. Adding the registry to your pull secret

Modify your the pull secret for your OpenShift Container Platform cluster to describe your local registry before you install an OpenShift Container Platform cluster in a restricted network.

Prerequisites

- You configured a mirror registry to use in your restricted network.

Procedure

Complete the following steps on the bastion host:

1. Download your **registry.redhat.io** pull secret from the Pull Secret page on the Red Hat OpenShift Cluster Manager site.

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)
```

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

1. For **bastion_host_name**, specify the registry domain name that you specified in your certificate, and for **<local_registry_host_port>**, specify the port that your mirror registry
2. For **<credentials>**, specify the base64-encoded user name and password for the mirror registry that you generated.

The file resembles 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"
    },
    "<local_registry_host_name>:<local_registry_host_port>": {
      "auth": "<credentials>",
      "email": "you@example.com"
    },
    "registry.redhat.io": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    }
  }
}
```

3.3.5. Mirroring the OpenShift Container Platform image repository

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

Prerequisites

- 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 bastion 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:

```
$ export OCP_RELEASE=<release_version> 1
$ export LOCAL_REGISTRY='<local_registry_host_name>:<local_registry_host_port>' 2
$ export LOCAL_REPOSITORY='<repository_name>' 3
$ export PRODUCT_REPO='openshift-release-dev' 4
$ export LOCAL_SECRET_JSON='<path_to_pull_secret>' 5
$ export RELEASE_NAME="ocp-release" 6
```

1. For **<release_version>**, specify the tag that corresponds to the version of OpenShift Container Platform to install for your architecture, such as **4.3.0-x86_64**.
2. 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.
3. For **<repository_name>**, specify the name of the repository to create in your registry, such as **ocp4/openshift4**.
4. The repository to mirror. For a production release, you must specify **openshift-release-dev**.
5. For **<path_to_pull_secret>**, specify the absolute path to and file name of the pull secret for your mirror registry that you created.
6. The release mirror. For a production release, you must specify **ocp-release**.

3. Mirror the repository:

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

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

4. 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.

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

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

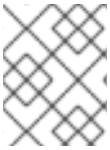
IMPORTANT

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.

3.3.6. Using Samples Operator imagestreams with alternate or mirrored registries

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



NOTE

The **cli**, **installer**, **must-gather**, and **tests** imagestreams, while part of the install payload, are not managed by the 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:

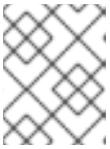
```
$ oc image mirror registry.redhat.io/rhscl/ruby-25-rhel7:latest ${MIRROR_ADDR}/rhscl/ruby-
25-rhel7:latest
```

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

```
$ oc create configmap registry-config --from-
file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
{"name":"registry-config"}}}' --type=merge
```

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

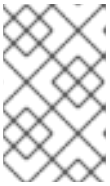
```
$ oc get configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```



NOTE

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

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

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).

Resource name	Description
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.
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 4.3, 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.

3.5. 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.

Procedure

1. Whitelist 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**

2. Whitelist 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
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.
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.

Cloud	URL	Function
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. Whitelist the following URLs:

URL	Function
mirror.openshift.com	Required to access mirrored installation content and images
***.cloudfront.net**	Required by the Quay CDN to deliver the Quay.io images that the cluster requires
***.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