OpenShift Container Platform 4.13

Installation configuration

Cluster-wide configuration during installations

Last Updated: 2024-12-13

OpenShift Container Platform 4.13 Installation configuration

Cluster-wide configuration during installations

Legal Notice

Abstract

This document describes how to perform initial OpenShift Container Platform cluster configuration.

Table of Contents

CHAPTER 1. CUSTOMIZING NODES

OpenShift Container Platform supports both cluster-wide and per-machine configuration via Ignition, which allows arbitrary partitioning and file content changes to the operating system. In general, if a configuration file is documented in Red Hat Enterprise Linux (RHEL), then modifying it via Ignition is supported.

There are two ways to deploy machine config changes:

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

Additionally, modifying the reference config, such as the Ignition config that is passed to **coreos-installer** when installing bare-metal nodes allows per-machine configuration. These changes are currently not visible to the Machine Config Operator.

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

1.1. CREATING MACHINE CONFIGS WITH BUTANE

Machine configs are used to configure control plane and worker machines by instructing machines how to create users and file systems, set up the network, install systemd units, and more.

Because modifying machine configs can be difficult, you can use Butane configs to create machine configs for you, thereby making node configuration much easier.

1.1.1. About Butane

Butane is a command-line utility that OpenShift Container Platform uses to provide convenient, short-hand syntax for writing machine configs, as well as for performing additional validation of machine configs. The format of the Butane config file that Butane accepts is defined in the OpenShift Butane config spec.

1.1.2. Installing Butane

You can install the Butane tool (**butane**) to create OpenShift Container Platform machine configs from a command-line interface. You can install **butane** on Linux, Windows, or macOS by downloading the corresponding binary file.

TIP

Butane releases are backwards-compatible with older releases and with the Fedora CoreOS Config Transpiler (FCCT).

Procedure

1. Navigate to the Butane image download page at https://mirror.openshift.com/pub/openshift-v4/clients/butane/.
2. Get the **butane** binary:
    a. For the newest version of Butane, save the latest **butane** image to your current directory:

```
$ curl https://mirror.openshift.com/pub/openshift-v4/clients/butane/latest/butane --output
butane
```

b. Optional: For a specific type of architecture you are installing Butane on, such as aarch64 or ppc64le, indicate the appropriate URL. For example:

```
$ curl https://mirror.openshift.com/pub/openshift-v4/clients/butane/latest/butane-aarch64
--output butane
```

3. Make the downloaded binary file executable:

```
$ chmod +x butane
```

4. Move the **butane** binary file to a directory on your **PATH**.
To check your **PATH**, open a terminal and execute the following command:

```
$ echo $PATH
```

Verification steps

- You can now use the Butane tool by running the **butane** command:

```
$ butane <butane_file>
```

1.1.3. Creating a MachineConfig object by using Butane

You can use Butane to produce a **MachineConfig** object so that you can configure worker or control plane nodes at installation time or via the Machine Config Operator.

Prerequisites

- You have installed the **butane** utility.

Procedure

1. Create a Butane config file. The following example creates a file named **99-worker-custom.bu** that configures the system console to show kernel debug messages and specifies custom settings for the chrony time service:

```
variant: openshift
version: 4.13.0
metadata:
  name: 99-worker-custom
  labels:
    machineconfiguration.openshift.io/role: worker
openshift:
  kernel_arguments:
    - loglevel=7
storage:
  files:
    - path: /etc/chrony.conf
      mode: 0644
```

```
    overwrite: true
    contents:
      inline: |
        pool 0.rhel.pool.ntp.org iburst
        driftfile /var/lib/chrony/drift
        makestep 1.0 3
        rtcsync
        logdir /var/log/chrony
```

NOTE

The **99-worker-custom.bu** file is set to create a machine config for worker nodes. To deploy on control plane 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.

2. Create a **MachineConfig** object by giving Butane the file that you created in the previous step:

```
$ butane 99-worker-custom.bu -o ./99-worker-custom.yaml
```

   A **MachineConfig** object YAML file is created for you to finish configuring your machines.

3. Save the Butane config in case you need to update the **MachineConfig** object in the future.

4. If the cluster is not running yet, generate manifest files and add the **MachineConfig** object YAML file to the **openshift** directory. If the cluster is already running, apply the file as follows:

```
$ oc create -f 99-worker-custom.yaml
```

Additional resources

- Adding kernel modules to nodes
- Encrypting and mirroring disks during installation

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

WARNING

Disabling SELinux on RHCOS in production is not supported. Once SELinux has been disabled on a node, it must be re-provisioned before re-inclusion in a production cluster.

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. Change to the directory that contains the installation program and 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 control plane 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 control plane 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.

1.3. 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.

1.3.1. 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 required 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:

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

a. 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.

1.3.2. 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** object 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.

1.3.2.1. Provision kernel modules via a MachineConfig object

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

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 needed to build the software:

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

4. Create a directory to host the kernel module and tooling:

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

5. 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
```

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

7. 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/
```

8. Clone the fakeroot directory, replacing any symbolic links with copies of their targets, by running the following command:

```
$ cd .. && rm -rf kmod-tree && cp -Lpr ${FAKEROOT} kmod-tree
```

9. Create a Butane config file, **99-simple-kmod.bu**, that embeds the kernel module tree and enables the systemd service.

NOTE

See "Creating machine configs with Butane" for information about Butane.

```
variant: openshift
version: 4.13.0
metadata:
  name: 99-simple-kmod
  labels:
    machineconfiguration.openshift.io/role: worker (1)
storage:
  trees:
    - local: kmod-tree
systemd:
  units:
    - name: kmods-via-containers@simple-kmod.service
      enabled: true
```

(1) To deploy on control plane nodes, change **worker** to **master**. To deploy on both control plane and worker nodes, perform the remainder of these instructions once for each node type.

10. Use Butane to generate a machine config YAML file, **99-simple-kmod.yaml**, containing the files and configuration to be delivered:

```
$ butane 99-simple-kmod.bu --files-dir . -o 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
```

1.4. ENCRYPTING AND MIRRORING DISKS DURING INSTALLATION

During an OpenShift Container Platform installation, you can enable boot disk encryption and mirroring on the cluster nodes.

1.4.1. About disk encryption

You can enable encryption for the boot disks on the control plane and compute nodes at installation time. OpenShift Container Platform supports the Trusted Platform Module (TPM) v2 and Tang encryption modes.

TPM v2

This is the preferred mode. TPM v2 stores passphrases in a secure cryptoprocessor on the server. You can use this mode to prevent decryption of the boot disk data on a cluster node if the disk is removed from the server.

Tang

Tang and Clevis are server and client components that enable network-bound disk encryption (NBDE). You can bind the boot disk data on your cluster nodes to one or more Tang servers. This prevents decryption of the data unless the nodes are on a secure network where the Tang servers are accessible. Clevis is an automated decryption framework used to implement decryption on the client side.

IMPORTANT

The use of the Tang encryption mode to encrypt your disks is only supported for bare metal and vSphere installations on user-provisioned infrastructure.

In earlier versions of Red Hat Enterprise Linux CoreOS (RHCOS), disk encryption was configured by specifying **/etc/clevis.json** in the Ignition config. That file is not supported in clusters created with OpenShift Container Platform 4.7 or later. Configure disk encryption by using the following procedure.

When the TPM v2 or Tang encryption modes are enabled, the RHCOS boot disks are encrypted using the LUKS2 format.

This feature:

- Is available for installer-provisioned infrastructure, user-provisioned infrastructure, and Assisted Installer deployments
- For Assisted installer deployments:
  - Each cluster can only have a single encryption method, Tang or TPM
  - Encryption can be enabled on some or all nodes
  - There is no Tang threshold; all servers must be valid and operational
  - Encryption applies to the installation disks only, not to the workload disks
- Is supported on Red Hat Enterprise Linux CoreOS (RHCOS) systems only
- Sets up disk encryption during the manifest installation phase, encrypting all data written to disk, from first boot forward
- Requires no user intervention for providing passphrases
- Uses AES-256-XTS encryption

1.4.1.1. Configuring an encryption threshold

In OpenShift Container Platform, you can specify a requirement for more than one Tang server. You can also configure the TPM v2 and Tang encryption modes simultaneously. This enables boot disk data decryption only if the TPM secure cryptoprocessor is present and the Tang servers are accessible over a secure network.

You can use the **threshold** attribute in your Butane configuration to define the minimum number of

TPM v2 and Tang encryption conditions required for decryption to occur. The threshold is met when the stated value is reached through any combination of the declared conditions. For example, the **threshold** value of **2** in the following configuration can be reached by accessing the two Tang servers, or by accessing the TPM secure cryptoprocessor and one of the Tang servers:

Example Butane configuration for disk encryption

```
variant: openshift
version: 4.13.0
metadata:
  name: worker-storage
  labels:
    machineconfiguration.openshift.io/role: worker
boot_device:
  layout: x86_64 (1)
  luks:
    tpm2: true (2)
    tang: (3)
      - url: http://tang1.example.com:7500
        thumbprint: jwGN5tRFK-kF6pIX89ssF3khxxX
      - url: http://tang2.example.com:7500
        thumbprint: VCJsvZFjBSIHSldw78rOrq7h2ZF
    threshold: 2 (4)
openshift:
  fips: true (5)
```

(1) Set this field to the instruction set architecture of the cluster nodes. Some examples include, **x86_64**, **aarch64**, or **ppc64le**.

(2) Include this field if you want to use a Trusted Platform Module (TPM) to encrypt the root file system.

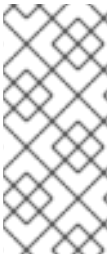
(3) Include this section if you want to use one or more Tang servers.

(4) Specify the minimum number of TPM v2 and Tang encryption conditions required for decryption to occur.

(5) OpenShift Container Platform 4.13 is based on Red Hat Enterprise Linux (RHEL) 9.2. RHEL 9.2 cryptographic modules have not yet been submitted for FIPS validation. For more information, see "About this release" in the 4.13 *OpenShift Container Platform Release Notes*.

IMPORTANT

The default **threshold** value is **1**. If you include multiple encryption conditions in your configuration but do not specify a threshold, decryption can occur if any of the conditions are met.



NOTE

If you require TPM v2 *and* Tang for decryption, the value of the **threshold** attribute must equal the total number of stated Tang servers plus one. If the **threshold** value is lower, it is possible to reach the threshold value by using a single encryption mode. For example, if you set **tpm2** to **true** and specify two Tang servers, a threshold of **2** can be met by accessing the two Tang servers, even if the TPM secure cryptoprocessor is not available.

1.4.2. About disk mirroring

During OpenShift Container Platform installation on control plane and worker nodes, you can enable mirroring of the boot and other disks to two or more redundant storage devices. A node continues to function after storage device failure provided one device remains available.

Mirroring does not support replacement of a failed disk. Reprovision the node to restore the mirror to a pristine, non-degraded state.

NOTE

For user-provisioned infrastructure deployments, mirroring is available only on RHCOS systems. Support for mirroring is available on **x86_64** nodes booted with BIOS or UEFI and on **ppc64le** nodes.

1.4.3. Configuring disk encryption and mirroring

You can enable and configure encryption and mirroring during an OpenShift Container Platform installation.

Prerequisites

- You have downloaded the OpenShift Container Platform installation program on your installation node.
- You installed Butane on your installation node.

NOTE

Butane is a command-line utility that OpenShift Container Platform uses to offer convenient, short-hand syntax for writing and validating machine configs. For more information, see "Creating machine configs with Butane".

- You have access to a Red Hat Enterprise Linux (RHEL) 8 machine that can be used to generate a thumbprint of the Tang exchange key.

Procedure

1. If you want to use TPM v2 to encrypt your cluster, check to see if TPM v2 encryption needs to be enabled in the host firmware for each node. This is required on most Dell systems. Check the manual for your specific system.
2. If you want to use Tang to encrypt your cluster, follow these preparatory steps:
   a. Set up a Tang server or access an existing one. See Network-bound disk encryption for instructions.

b. Install the **clevis** package on a RHEL 8 machine, if it is not already installed:

```
$ sudo yum install clevis
```

c. On the RHEL 8 machine, run the following command to generate a thumbprint of the exchange key. Replace **http://tang.example.com:7500** with the URL of your Tang server:

```
$ clevis-encrypt-tang '{"url":"http://tang.example.com:7500"}' < /dev/null > /dev/null 1
```

1 In this example, **tangd.socket** is listening on port **7500** on the Tang server.

NOTE

The **clevis-encrypt-tang** command generates a thumbprint of the exchange key. No data passes to the encryption command during this step; **/dev/null** exists here as an input instead of plain text. The encrypted output is also sent to **/dev/null**, because it is not required for this procedure.

Example output

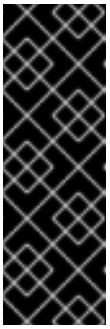
```
The advertisement contains the following signing keys:

PLjNyRdGw03zlRoGjQYMahSZGu9 1
```

1 The thumbprint of the exchange key.

When the **Do you wish to trust these keys? [ynYN]** prompt displays, type **Y**.

d. If the nodes are configured with static IP addressing, run **coreos-installer iso customize --dest-karg-append** or use the **coreos-installer --append-karg** option when installing RHCOS nodes to set the IP address of the installed system. Append the **ip=** and other arguments needed for your network.



IMPORTANT

Some methods for configuring static IPs do not affect the initramfs after the first boot and will not work with Tang encryption. These include the **coreos-installer --copy-network** option, the **coreos-installer iso customize --network-keyfile** option, and the **coreos-installer pxe customize --network-keyfile** option, as well as adding **ip=** arguments to the kernel command line of the live ISO or PXE image during installation. Incorrect static IP configuration causes the second boot of the node to fail.

3. On your installation node, change to the directory that contains the installation program and generate the Kubernetes manifests for the cluster:

```
$ ./openshift-install create manifests --dir <installation_directory> 1
```

1 Replace **<installation_directory>** with the path to the directory that you want to store the installation files in.

4. Create a Butane config that configures disk encryption, mirroring, or both. For example, to configure storage for compute nodes, create a **$HOME/clusterconfig/worker-storage.bu** file.

Butane config example for a boot device

```
variant: openshift
version: 4.13.0
metadata:
  name: worker-storage (1)
  labels:
    machineconfiguration.openshift.io/role: worker (2)
boot_device:
  layout: x86_64 (3)
  luks: (4)
    tpm2: true (5)
    tang: (6)
      - url: http://tang.example.com:7500 (7)
        thumbprint: PLjNyRdGw03zlRoGjQYMahSZGu9 (8)
    threshold: 1 (9)
  mirror: (10)
    devices: (11)
      - /dev/sda
      - /dev/sdb
openshift:
  fips: true (12)
```

(1) (2) For control plane configurations, replace **worker** with **master** in both of these locations.

(3) Set this field to the instruction set architecture of the cluster nodes. Some examples include, **x86_64**, **aarch64**, or **ppc64le**.

(4) Include this section if you want to encrypt the root file system. For more details, see "About disk encryption".

(5) Include this field if you want to use a Trusted Platform Module (TPM) to encrypt the root file system.

(6) Include this section if you want to use one or more Tang servers.

(7) Specify the URL of a Tang server. In this example, **tangd.socket** is listening on port **7500** on the Tang server.

(8) Specify the exchange key thumbprint, which was generated in a preceding step.

(9) Specify the minimum number of TPM v2 and Tang encryption conditions that must be met for decryption to occur. The default value is **1**. For more information about this topic, see "Configuring an encryption threshold".

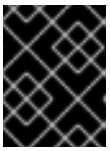
(10) Include this section if you want to mirror the boot disk. For more details, see "About disk mirroring".

(11) List all disk devices that should be included in the boot disk mirror, including the disk that RHCOS will be installed onto.

12 Include this directive to enable FIPS mode on your cluster.

IMPORTANT

OpenShift Container Platform 4.13 is based on Red Hat Enterprise Linux (RHEL) 9.2. RHEL 9.2 cryptographic modules have not yet been submitted for FIPS validation. For more information, see "About this release" in the 4.13 *OpenShift Container Platform Release Notes*.



IMPORTANT

If you are configuring nodes to use both disk encryption and mirroring, both features must be configured in the same Butane config.

5. Create a control plane or compute node manifest from the corresponding Butane config and save it to the **<installation_directory>/openshift** directory. For example, to create a manifest for the compute nodes, run the following command:

```
$ butane $HOME/clusterconfig/worker-storage.bu -o <installation_directory>/openshift/99-worker-storage.yaml
```

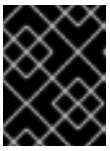
   Repeat this step for each node type that requires disk encryption or mirroring.

6. Save the Butane configs in case you need to update the manifests in the future.

7. Continue with the remainder of the OpenShift Container Platform installation.

TIP

You can monitor the console log on the RHCOS nodes during installation for error messages relating to disk encryption or mirroring.



IMPORTANT

If you configure additional data partitions, they will not be encrypted unless encryption is explicitly requested.

Verification

After installing OpenShift Container Platform, you can verify if boot disk encryption or mirroring is enabled on the cluster nodes.

1. From the installation host, access a cluster node by using a debug pod:

   a. Start a debug pod for the node, for example:

```
$ oc debug node/compute-1
```

   b. Set **/host** as the root directory within the debug shell. The debug pod mounts the root file system of the node in **/host** within the pod. By changing the root directory to **/host**, you can run binaries contained in the executable paths on the node:

```
# chroot /host
```

NOTE

OpenShift Container Platform cluster nodes running Red Hat Enterprise Linux CoreOS (RHCOS) are immutable and rely on Operators to apply cluster changes. Accessing cluster nodes using SSH is not recommended. However, if the OpenShift Container Platform API is not available, or **kubelet** is not properly functioning on the target node, **oc** operations will be impacted. In such situations, it is possible to access nodes using **ssh core@<node>.<cluster_name>.<base_domain>** instead.

2. If you configured boot disk encryption, verify if it is enabled:

   a. From the debug shell, review the status of the root mapping on the node:

   ```
   # cryptsetup status root
   ```

   Example output

   ```
   /dev/mapper/root is active and is in use.
     type:    LUKS2 1
     cipher:  aes-xts-plain64 2
     keysize: 512 bits
     key location: keyring
     device:  /dev/sda4 3
     sector size:  512
     offset:  32768 sectors
     size:    15683456 sectors
     mode:    read/write
   ```

   1 The encryption format. When the TPM v2 or Tang encryption modes are enabled, the RHCOS boot disks are encrypted using the LUKS2 format.

   2 The encryption algorithm used to encrypt the LUKS2 volume.

   3 The device that contains the encrypted LUKS2 volume. If mirroring is enabled, the value will represent a software mirror device, for example **/dev/md126**.

   b. List the Clevis plugins that are bound to the encrypted device:

   ```
   # clevis luks list -d /dev/sda4 1
   ```

   1 Specify the device that is listed in the **device** field in the output of the preceding step.

   Example output

   ```
   1: sss '{"t":1,"pins":{"tang":[{"url":"http://tang.example.com:7500"}]}}' 1
   ```

   1 In the example output, the Tang plugin is used by the Shamir's Secret Sharing (SSS) Clevis plugin for the **/dev/sda4** device.

3. If you configured mirroring, verify if it is enabled:

a. From the debug shell, list the software RAID devices on the node:

```
# cat /proc/mdstat
```

Example output

```
Personalities : [raid1]
md126 : active raid1 sdb3[1] sda3[0] 1
  393152 blocks super 1.0 [2/2] [UU]

md127 : active raid1 sda4[0] sdb4[1] 2
  51869632 blocks super 1.2 [2/2] [UU]

unused devices: <none>
```

1 The **/dev/md126** software RAID mirror device uses the **/dev/sda3** and **/dev/sdb3** disk devices on the cluster node.

2 The **/dev/md127** software RAID mirror device uses the **/dev/sda4** and **/dev/sdb4** disk devices on the cluster node.

b. Review the details of each of the software RAID devices listed in the output of the preceding command. The following example lists the details of the **/dev/md126** device:

```
# mdadm --detail /dev/md126
```

Example output

```
/dev/md126:
           Version : 1.0
     Creation Time : Wed Jul  7 11:07:36 2021
        Raid Level : raid1 1
        Array Size : 393152 (383.94 MiB 402.59 MB)
     Used Dev Size : 393152 (383.94 MiB 402.59 MB)
      Raid Devices : 2
     Total Devices : 2
       Persistence : Superblock is persistent

       Update Time : Wed Jul  7 11:18:24 2021
             State : clean 2
    Active Devices : 2 3
   Working Devices : 2 4
    Failed Devices : 0 5
     Spare Devices : 0

Consistency Policy : resync

              Name : any:md-boot 6
              UUID : ccfa3801:c520e0b5:2bee2755:69043055
            Events : 19
```

```
Number   Major   Minor   RaidDevice State
   0     252       3        0      active sync   /dev/sda3 7
   1     252      19        1      active sync   /dev/sdb3 8
```

1 Specifies the RAID level of the device. **raid1** indicates RAID 1 disk mirroring.

2 Specifies the state of the RAID device.

3 4 States the number of underlying disk devices that are active and working.

5 States the number of underlying disk devices that are in a failed state.

6 The name of the software RAID device.

7 8 Provides information about the underlying disk devices used by the software RAID device.

c. List the file systems mounted on the software RAID devices:

```
# mount | grep /dev/md
```

Example output

```
/dev/md127 on / type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /etc type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /usr type xfs
(ro,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /sysroot type xfs
(ro,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /var type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /var/lib/containers/storage/overlay type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /var/lib/kubelet/pods/e5054ed5-f882-4d14-b599-99c050d4e0c0/volume-
subpaths/etc/tuned/1 type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /var/lib/kubelet/pods/e5054ed5-f882-4d14-b599-99c050d4e0c0/volume-
subpaths/etc/tuned/2 type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /var/lib/kubelet/pods/e5054ed5-f882-4d14-b599-99c050d4e0c0/volume-
subpaths/etc/tuned/3 type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /var/lib/kubelet/pods/e5054ed5-f882-4d14-b599-99c050d4e0c0/volume-
subpaths/etc/tuned/4 type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md127 on /var/lib/kubelet/pods/e5054ed5-f882-4d14-b599-99c050d4e0c0/volume-
subpaths/etc/tuned/5 type xfs
(rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota)
/dev/md126 on /boot type ext4 (rw,relatime,seclabel)
```

In the example output, the **/boot** file system is mounted on the **/dev/md126** software RAID device and the root file system is mounted on **/dev/md127**.

4. Repeat the verification steps for each OpenShift Container Platform node type.

Additional resources

- For more information about the TPM v2 and Tang encryption modes, see Configuring automated unlocking of encrypted volumes using policy-based decryption.

1.4.4. Configuring a RAID-enabled data volume

You can enable software RAID partitioning to provide an external data volume. OpenShift Container Platform supports RAID 0, RAID 1, RAID 4, RAID 5, RAID 6, and RAID 10 for data protection and fault tolerance. See "About disk mirroring" for more details.

Prerequisites

- You have downloaded the OpenShift Container Platform installation program on your installation node.
- You have installed Butane on your installation node.

NOTE

Butane is a command-line utility that OpenShift Container Platform uses to provide convenient, short-hand syntax for writing machine configs, as well as for performing additional validation of machine configs. For more information, see the *Creating machine configs with Butane* section.

Procedure

1. Create a Butane config that configures a data volume by using software RAID.
   - To configure a data volume with RAID 1 on the same disks that are used for a mirrored boot disk, create a **$HOME/clusterconfig/raid1-storage.bu** file, for example:

     RAID 1 on mirrored boot disk

```
variant: openshift
version: 4.13.0
metadata:
  name: raid1-storage
  labels:
    machineconfiguration.openshift.io/role: worker
boot_device:
  mirror:
    devices:
      - /dev/disk/by-id/scsi-3600508b400105e210000900000490000
      - /dev/disk/by-id/scsi-SSEAGATE_ST373453LW_3HW1RHM6
storage:
  disks:
    - device: /dev/disk/by-id/scsi-3600508b400105e210000900000490000
      partitions:
        - label: root-1
          size_mib: 25000 (1)
        - label: var-1
    - device: /dev/disk/by-id/scsi-SSEAGATE_ST373453LW_3HW1RHM6
```

```
      partitions:
        - label: root-2
          size_mib: 25000 2
        - label: var-2
    raid:
      - name: md-var
        level: raid1
        devices:
          - /dev/disk/by-partlabel/var-1
          - /dev/disk/by-partlabel/var-2
    filesystems:
      - device: /dev/md/md-var
        path: /var
        format: xfs
        wipe_filesystem: true
        with_mount_unit: true
```

1 2 When adding a data partition to the boot disk, a minimum value of 25000 mebibytes is recommended. If no value is specified, or if the specified value is smaller than the recommended minimum, the resulting root file system will be too small, and future reinstalls of RHCOS might overwrite the beginning of the data partition.

- To configure a data volume with RAID 1 on secondary disks, create a **$HOME/clusterconfig/raid1-alt-storage.bu** file, for example:

  RAID 1 on secondary disks

```
variant: openshift
version: 4.13.0
metadata:
  name: raid1-alt-storage
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  disks:
    - device: /dev/sdc
      wipe_table: true
      partitions:
        - label: data-1
    - device: /dev/sdd
      wipe_table: true
      partitions:
        - label: data-2
  raid:
    - name: md-var-lib-containers
      level: raid1
      devices:
        - /dev/disk/by-partlabel/data-1
        - /dev/disk/by-partlabel/data-2
  filesystems:
    - device: /dev/md/md-var-lib-containers
      path: /var/lib/containers
      format: xfs
      wipe_filesystem: true
      with_mount_unit: true
```

2. Create a RAID manifest from the Butane config you created in the previous step and save it to the **<installation_directory>/openshift** directory. For example, to create a manifest for the compute nodes, run the following command:

```
$ butane $HOME/clusterconfig/<butane_config>.bu -o
<installation_directory>/openshift/<manifest_name>.yaml 1
```

1 Replace **<butane_config>** and **<manifest_name>** with the file names from the previous step. For example, **raid1-alt-storage.bu** and **raid1-alt-storage.yaml** for secondary disks.

3. Save the Butane config in case you need to update the manifest in the future.

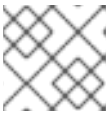
4. Continue with the remainder of the OpenShift Container Platform installation.

1.5. 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 machine config.

Procedure

1. Create a Butane config including the contents of the **chrony.conf** file. For example, to configure chrony on worker nodes, create a **99-worker-chrony.bu** file.



NOTE

See "Creating machine configs with Butane" for information about Butane.

```
variant: openshift
version: 4.13.0
metadata:
  name: 99-worker-chrony 1
  labels:
    machineconfiguration.openshift.io/role: worker 2
storage:
  files:
  - path: /etc/chrony.conf
    mode: 0644 3
    overwrite: true
    contents:
      inline: |
        pool 0.rhel.pool.ntp.org iburst 4
        driftfile /var/lib/chrony/drift
        makestep 1.0 3
        rtcsync
        logdir /var/log/chrony
```

1 2 On control plane nodes, substitute **master** for **worker** in both of these locations.

3 Specify an octal value mode for the **mode** field in the machine config file. After creating the file and applying the changes, the **mode** is converted to a decimal value. You can check

the YAML file with the command **oc get mc <mc-name> -o yaml**.

4 Specify any valid, reachable time source, such as the one provided by your DHCP server. 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**.

2. Use Butane to generate a **MachineConfig** object file, **99-worker-chrony.yaml**, containing the configuration to be delivered to the nodes:

```
$ butane 99-worker-chrony.bu -o 99-worker-chrony.yaml
```

3. Apply the configurations in one of two ways:

- If the cluster is not running yet, after you generate manifest files, add the **MachineConfig** object file to the **<installation_directory>/openshift** directory, and then continue to create the cluster.
- If the cluster is already running, apply the file:

```
$ oc apply -f ./99-worker-chrony.yaml
```

1.6. ADDITIONAL RESOURCES

- For information on Butane, see Creating machine configs with Butane .

CHAPTER 2. 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.

2.1. 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 compared to worker nodes.

NOTE

If your environment has a dedicated load balancer in front of your OpenShift Container Platform cluster, review the allowlists between your firewall and load balancer to prevent unwanted network restrictions to your cluster.

Procedure

1. Allowlist the following registry URLs:

URL	Port	Function
registry.redhat.io	443	Provides core container images
access.redhat.com	443	Hosts a signature store that a container client requires for verifying images pulled from **registry.access.redhat.com**. In a firewall environment, ensure that this resource is on the allowlist.
registry.access.redhat.com	443	Hosts all the container images that are stored on the Red Hat Ecosystem Catalog, including core container images.
quay.io	443	Provides core container images
cdn.quay.io	443	Provides core container images
cdn01.quay.io	443	Provides core container images
cdn02.quay.io	443	Provides core container images
cdn03.quay.io	443	Provides core container images
cdn04.quay.io	443	Provides core container images

URL	Port	Function
cdn05.quay.io	443	Provides core container images
cdn06.quay.io	443	Provides core container images
sso.redhat.com	443	The **https://console.redhat.com** site uses authentication from **sso.redhat.com**

- You can use the wildcards ***.quay.io** and ***.openshiftapps.com** instead of **cdn.quay.io** and **cdn0[1-6].quay.io** in your allowlist.
- You can use the wildcard ***.access.redhat.com** to simplify the configuration and ensure that all subdomains, including **registry.access.redhat.com**, are allowed.
- When you add a site, such as **quay.io**, to your allowlist, do not add a wildcard entry, such as ***.quay.io**, to your denylist. In most cases, image registries use a content delivery network (CDN) to serve images. If a firewall blocks access, image downloads are denied when the initial download request redirects to a hostname such as **cdn01.quay.io**.

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	Port	Function
cert-api.access.redhat.com	443	Required for Telemetry
api.access.redhat.com	443	Required for Telemetry
infogw.api.openshift.com	443	Required for Telemetry
console.redhat.com	443	Required for Telemetry and for **insights-operator**

4. If you use Alibaba Cloud, 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	Port	Function
Alibaba	***.aliyuncs.com**	443	Required to access Alibaba Cloud services and resources. Review the Alibaba endpoints_config.go file to determine the exact endpoints to allow for the regions that you use.

Cloud	URL	Port	Function
AWS	**aws.amazon.com**	443	Used to install and manage clusters in an AWS environment.
	***.amazonaws.com** Alternatively, if you choose to not use a wildcard for AWS APIs, you must allowlist the following URLs:	443	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.
	ec2.amazonaws.com	443	Used to install and manage clusters in an AWS environment.
	events.amazonaws.com	443	Used to install and manage clusters in an AWS environment.
	iam.amazonaws.com	443	Used to install and manage clusters in an AWS environment.
	route53.amazonaws.com	443	Used to install and manage clusters in an AWS environment.
	***.s3.amazonaws.com**	443	Used to install and manage clusters in an AWS environment.
	***.s3.<aws_region>.amazonaws.com**	443	Used to install and manage clusters in an AWS environment.
	***.s3.dualstack.<aws_region>.amazonaws.com**	443	Used to install and manage clusters in an AWS environment.
	sts.amazonaws.com	443	Used to install and manage clusters in an AWS environment.
	sts.<aws_region>.amazonaws.com	443	Used to install and manage clusters in an AWS environment.
	tagging.us-east-1.amazonaws.com	443	Used to install and manage clusters in an AWS environment. This endpoint is always **us-east-1**, regardless of the region the cluster is deployed in.
	ec2.<aws_region>.amazonaws.com	443	Used to install and manage clusters in an AWS environment.
	elasticloadbalancing.<aws_region>.amazonaws.com	443	Used to install and manage clusters in an AWS environment.

Cloud	URL	Port	Function
	servicequotas.<aws_region>.amazonaws.com	443	Required. Used to confirm quotas for deploying the service.
	tagging.<aws_region>.amazonaws.com	443	Allows the assignment of metadata about AWS resources in the form of tags.
GCP	***.googleapis.com**	443	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	443	Required to access your GCP account.
Azure	**management.azure.com**	443	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.
	***.blob.core.windows.net**	443	Required to download Ignition files.
	login.microsoftonline.com	443	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	Port	Function
***.apps.<cluster_name>.<base_domain>**	443	Required to access the default cluster routes unless you set an ingress wildcard during installation.
api.openshift.com	443	Required both for your cluster token and to check if updates are available for the cluster.
console.redhat.com	443	Required for your cluster token.
mirror.openshift.com	443	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	Port	Function
quayio-production-s3.s3.amazonaws.com	443	Required to access Quay image content in AWS.
rhcos.mirror.openshift.com	443	Required to download Red Hat Enterprise Linux CoreOS (RHCOS) images.
sso.redhat.com	443	The **https://console.redhat.com** site uses authentication from **sso.redhat.com**
storage.googleapis.com/openshift-release	443	A source of release image signatures, although the Cluster Version Operator needs only a single functioning source.

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 hostname that is specified in the **spec.route.hostname** field of the **consoles.operator/cluster** object if the field is not empty.

6. Allowlist the following URLs for optional third-party content:

URL	Port	Function
registry.connect.redhat.com	443	Required for all third-party images and certified operators.
rhc4tp-prod-z8cxf-image-registry-us-east-1-evenkyleffocxqvofrk.s3.dualstack.us-east-1.amazonaws.com	443	Provides access to container images hosted on **registry.connect.redhat.com**
oso-rhc4tp-docker-registry.s3-us-west-2.amazonaws.com	443	Required for Sonatype Nexus, F5 Big IP operators.

7. If you use a default Red Hat Network Time Protocol (NTP) server allow the following URLs:
   - **1.rhel.pool.ntp.org**
   - **2.rhel.pool.ntp.org**
   - **3.rhel.pool.ntp.org**

NOTE

If you do not use a default Red Hat NTP server, verify the NTP server for your platform and allow it in your firewall.

CHAPTER 3. ENABLING LINUX CONTROL GROUP VERSION 2 (CGROUP V2)

By default, OpenShift Container Platform uses Linux control group version 1 (cgroup v1) in your cluster. You can enable Linux control group version 2 (cgroup v2) upon installation. Enabling cgroup v2 in OpenShift Container Platform disables all cgroup version 1 controllers and hierarchies in your cluster.

cgroup v2 is the next version of the Linux cgroup API. cgroup v2 offers several improvements over cgroup v1, including a unified hierarchy, safer sub-tree delegation, new features such as Pressure Stall Information, and enhanced resource management and isolation.

You can switch between cgroup v1 and cgroup v2, as needed, by editing the **node.config** object. For more information, see "Configuring the Linux cgroup on your nodes" in the "Additional resources" of this section.

3.1. ENABLING LINUX CGROUP V2 DURING INSTALLATION

You can enable Linux control group version 2 (cgroup v2) when you install a cluster by creating installation manifests.

Procedure

1. Create or edit the **node.config** object to specify the **v2** cgroup:

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
  name: cluster
spec:
  cgroupMode: "v2"
```

2. Proceed with the installation as usual.

Additional resources

- OpenShift Container Platform installation overview
- Configuring the Linux cgroup on your nodes