OpenShift Container Platform 4.16

Machine configuration

Managing and applying configuration and updates of the base operating system and container runtimes in OpenShift Container Platform

OpenShift Container Platform 4.16 Machine configuration

Managing and applying configuration and updates of the base operating system and container runtimes in OpenShift Container Platform

Legal Notice

Abstract

This document provides instructions for managing changes to systemd, CRI-O, Kubelet, the kernel, and other system features by using MachineConfig, KubeletConfig, and ContainerRuntimeConfig objects. In addition, image layering allows you to easily customize the underlying node operating system by layering additional images onto the base image of any of your cluster worker nodes.

Table of Contents

CHAPTER 1. MACHINE CONFIGURATION OVERVIEW

There are times when you need to make changes to the operating systems running on OpenShift Container Platform nodes. This can include changing settings for network time service, adding kernel arguments, or configuring journaling in a specific way.

Aside from a few specialized features, most changes to operating systems on OpenShift Container Platform nodes can be done by creating what are referred to as **MachineConfig** objects that are managed by the Machine Config Operator. For example, you can use the Machine Config Operator (MCO) and machine configs to manage update to systemd, CRI-O and kubelet, the kernel, Network Manager and other system features.

Tasks in this section describe how to use features of the Machine Config Operator to configure operating system features on OpenShift Container Platform nodes.

> **IMPORTANT**
>
> NetworkManager stores new network configurations to **/etc/NetworkManager/system-connections/** in a key file format.
>
> Previously, NetworkManager stored new network configurations to **/etc/sysconfig/network-scripts/** in the **ifcfg** format. Starting with RHEL 9.0, RHEL stores new network configurations at **/etc/NetworkManager/system-connections/** in a key file format. The connections configurations stored to **/etc/sysconfig/network-scripts/** in the old format still work uninterrupted. Modifications in existing profiles continue updating the older files.

1.1. ABOUT THE MACHINE CONFIG OPERATOR

OpenShift Container Platform 4.16 integrates both operating system and cluster management. Because the cluster manages its own updates, including updates to Red Hat Enterprise Linux CoreOS (RHCOS) on cluster nodes, OpenShift Container Platform provides an opinionated lifecycle management experience that simplifies the orchestration of node upgrades.

OpenShift Container Platform employs three daemon sets and controllers to simplify node management. These daemon sets orchestrate operating system updates and configuration changes to the hosts by using standard Kubernetes-style constructs. They include:

- The **machine-config-controller**, which coordinates machine upgrades from the control plane. It monitors all of the cluster nodes and orchestrates their configuration updates.

- The **machine-config-daemon** daemon set, which runs on each node in the cluster and updates a machine to configuration as defined by machine config and as instructed by the MachineConfigController. When the node detects a change, it drains off its pods, applies the update, and reboots. These changes come in the form of Ignition configuration files that apply the specified machine configuration and control kubelet configuration. The update itself is delivered in a container. This process is key to the success of managing OpenShift Container Platform and RHCOS updates together.

- The **machine-config-server** daemon set, which provides the Ignition config files to control plane nodes as they join the cluster.

The machine configuration is a subset of the Ignition configuration. The **machine-config-daemon** reads the machine configuration to see if it needs to do an OSTree update or if it must apply a series of systemd kubelet file changes, configuration changes, or other changes to the operating system or

OpenShift Container Platform configuration.

When you perform node management operations, you create or modify a **KubeletConfig** custom resource (CR).

> **IMPORTANT**
>
> When changes are made to a machine configuration, the Machine Config Operator (MCO) automatically reboots all corresponding nodes in order for the changes to take effect.
>
> You can mitigate the disruption caused by some machine config changes by using a node disruption policy. See *Understanding node restart behaviors after machine config changes* .
>
> Alternatively, you can prevent the nodes from automatically rebooting after machine configuration changes before making the changes. Pause the autoreboot process by setting the **spec.paused** field to **true** in the corresponding machine config pool. When paused, machine configuration changes are not applied until you set the **spec.paused** field to **false** and the nodes have rebooted into the new configuration.
>
> The following modifications do not trigger a node reboot:
>
> - When the MCO detects any of the following changes, it applies the update without draining or rebooting the node:
>
>   - Changes to the SSH key in the **spec.config.passwd.users.sshAuthorizedKeys** parameter of a machine config.
>
>   - Changes to the global pull secret or pull secret in the **openshift-config** namespace.
>
>   - Automatic rotation of the **/etc/kubernetes/kubelet-ca.crt** certificate authority (CA) by the Kubernetes API Server Operator.
>
> - When the MCO detects changes to the **/etc/containers/registries.conf** file, such as adding or editing an **ImageDigestMirrorSet**, **ImageTagMirrorSet**, or **ImageContentSourcePolicy** object, it drains the corresponding nodes, applies the changes, and uncordons the nodes. The node drain does not happen for the following changes:
>
>   - The addition of a registry with the **pull-from-mirror = "digest-only"** parameter set for each mirror.
>
>   - The addition of a mirror with the **pull-from-mirror = "digest-only"** parameter set in a registry.
>
>   - The addition of items to the **unqualified-search-registries** list.

There might be situations where the configuration on a node does not fully match what the currently-applied machine config specifies. This state is called *configuration drift*. The Machine Config Daemon (MCD) regularly checks the nodes for configuration drift. If the MCD detects configuration drift, the MCO marks the node **degraded** until an administrator corrects the node configuration. A degraded node is online and operational, but, it cannot be updated.

Additional resources

- About the OpenShift SDN network plugin

1.2. MACHINE CONFIG OVERVIEW

The Machine Config Operator (MCO) manages updates to systemd, CRI-O and Kubelet, the kernel, Network Manager and other system features. It also offers a **MachineConfig** CRD that can write configuration files onto the host (see machine-config-operator). Understanding what MCO does and how it interacts with other components is critical to making advanced, system-level changes to an OpenShift Container Platform cluster. Here are some things you should know about MCO, machine configs, and how they are used:

- Machine configs are processed alphabetically, in lexicographically increasing order, of their name. The render controller uses the first machine config in the list as the base and appends the rest to the base machine config.

- A machine config can make a specific change to a file or service on the operating system of each system representing a pool of OpenShift Container Platform nodes.

- MCO applies changes to operating systems in pools of machines. All OpenShift Container Platform clusters start with worker and control plane node pools. By adding more role labels, you can configure custom pools of nodes. For example, you can set up a custom pool of worker nodes that includes particular hardware features needed by an application. However, examples in this section focus on changes to the default pool types.

> **IMPORTANT**
>
> A node can have multiple labels applied that indicate its type, such as **master** or **worker**, however it can be a member of only a single machine config pool.

- After a machine config change, the MCO updates the affected nodes alphabetically by zone, based on the **topology.kubernetes.io/zone** label. If a zone has more than one node, the oldest nodes are updated first. For nodes that do not use zones, such as in bare metal deployments, the nodes are upgraded by age, with the oldest nodes updated first. The MCO updates the number of nodes as specified by the **maxUnavailable** field on the machine configuration pool at a time.

- Some machine configuration must be in place before OpenShift Container Platform is installed to disk. In most cases, this can be accomplished by creating a machine config that is injected directly into the OpenShift Container Platform installer process, instead of running as a postinstallation machine config. In other cases, you might need to do bare metal installation where you pass kernel arguments at OpenShift Container Platform installer startup, to do such things as setting per-node individual IP addresses or advanced disk partitioning.

- MCO manages items that are set in machine configs. Manual changes you do to your systems will not be overwritten by MCO, unless MCO is explicitly told to manage a conflicting file. In other words, MCO only makes specific updates you request, it does not claim control over the whole node.

- Manual changes to nodes are strongly discouraged. If you need to decommission a node and start a new one, those direct changes would be lost.

- MCO is only supported for writing to files in **/etc** and **/var** directories, although there are symbolic links to some directories that can be writeable by being symbolically linked to one of those areas. The **/opt** and **/usr/local** directories are examples.

- Ignition is the configuration format used in MachineConfigs. See the Ignition Configuration Specification v3.2.0 for details.

- Although Ignition config settings can be delivered directly at OpenShift Container Platform installation time, and are formatted in the same way that MCO delivers Ignition configs, MCO has no way of seeing what those original Ignition configs are. Therefore, you should wrap Ignition config settings into a machine config before deploying them.

- When a file managed by MCO changes outside of MCO, the Machine Config Daemon (MCD) sets the node as **degraded**. It will not overwrite the offending file, however, and should continue to operate in a **degraded** state.

- A key reason for using a machine config is that it will be applied when you spin up new nodes for a pool in your OpenShift Container Platform cluster. The **machine-api-operator** provisions a new machine and MCO configures it.

MCO uses Ignition as the configuration format. OpenShift Container Platform 4.6 moved from Ignition config specification version 2 to version 3.

1.2.1. What can you change with machine configs?

The kinds of components that MCO can change include:

- **config**: Create Ignition config objects (see the Ignition configuration specification) to do things like modify files, systemd services, and other features on OpenShift Container Platform machines, including:

  - **Configuration files**: Create or overwrite files in the **/var** or **/etc** directory.

  - **systemd units**: Create and set the status of a systemd service or add to an existing systemd service by dropping in additional settings.

  - **users and groups**: Change SSH keys in the passwd section postinstallation.

    IMPORTANT

    - Changing SSH keys by using a machine config is supported only for the **core** user.

    - Adding new users by using a machine config is not supported.

- **kernelArguments**: Add arguments to the kernel command line when OpenShift Container Platform nodes boot.

- **kernelType**: Optionally identify a non-standard kernel to use instead of the standard kernel. Use **realtime** to use the RT kernel (for RAN). This is only supported on select platforms. Use the **64k-pages** parameter to enable the 64k page size kernel. This setting is exclusive to machines with 64-bit ARM architectures.

- **fips**: Enable FIPS mode. FIPS should be set at installation-time setting and not a postinstallation procedure.

IMPORTANT

To enable FIPS mode for your cluster, you must run the installation program from a Red Hat Enterprise Linux (RHEL) computer configured to operate in FIPS mode. For more information about configuring FIPS mode on RHEL, see Installing the system in FIPS mode .

When running Red Hat Enterprise Linux (RHEL) or Red Hat Enterprise Linux CoreOS (RHCOS) booted in FIPS mode, OpenShift Container Platform core components use the RHEL cryptographic libraries that have been submitted to NIST for FIPS 140-2/140-3 Validation on only the x86_64, ppc64le, and s390x architectures.

- **extensions**: Extend RHCOS features by adding selected pre-packaged software. For this feature, available extensions include usbguard and kernel modules.

- **Custom resources (for ContainerRuntime and Kubelet)**: Outside of machine configs, MCO manages two special custom resources for modifying CRI-O container runtime settings (**ContainerRuntime** CR) and the Kubelet service (**Kubelet** CR).

The MCO is not the only Operator that can change operating system components on OpenShift Container Platform nodes. Other Operators can modify operating system-level features as well. One example is the Node Tuning Operator, which allows you to do node-level tuning through Tuned daemon profiles.

Tasks for the MCO configuration that can be done after installation are included in the following procedures. See descriptions of RHCOS bare metal installation for system configuration tasks that must be done during or before OpenShift Container Platform installation. By default, many of the changes you make with the MCO require a reboot.

The following modifications do not trigger a node reboot:

- When the MCO detects any of the following changes, it applies the update without draining or rebooting the node:

  - Changes to the SSH key in the **spec.config.passwd.users.sshAuthorizedKeys** parameter of a machine config.

  - Changes to the global pull secret or pull secret in the **openshift-config** namespace.

  - Automatic rotation of the **/etc/kubernetes/kubelet-ca.crt** certificate authority (CA) by the Kubernetes API Server Operator.

- When the MCO detects changes to the **/etc/containers/registries.conf** file, such as adding or editing an **ImageDigestMirrorSet**, **ImageTagMirrorSet**, or **ImageContentSourcePolicy** object, it drains the corresponding nodes, applies the changes, and uncordons the nodes. The node drain does not happen for the following changes:

  - The addition of a registry with the **pull-from-mirror = "digest-only"** parameter set for each mirror.

  - The addition of a mirror with the **pull-from-mirror = "digest-only"** parameter set in a registry.

  - The addition of items to the **unqualified-search-registries** list.

In other cases, you can mitigate the disruption to your workload when the MCO makes changes by using *node disruption policies*. For information, see *Understanding node restart behaviors after machine config changes.*

There might be situations where the configuration on a node does not fully match what the currently-applied machine config specifies. This state is called *configuration drift*. The Machine Config Daemon (MCD) regularly checks the nodes for configuration drift. If the MCD detects configuration drift, the MCO marks the node **degraded** until an administrator corrects the node configuration. A degraded node is online and operational, but, it cannot be updated. For more information on configuration drift, see *Understanding configuration drift detection*.

1.3. UNDERSTANDING THE MACHINE CONFIG OPERATOR NODE DRAIN BEHAVIOR

When you use a machine config to change a system feature, such as adding new config files, modifying systemd units or kernel arguments, or updating SSH keys, the Machine Config Operator (MCO) applies those changes and ensures that each node is in the desired configuration state.

After you make the changes, the MCO generates a new rendered machine config. In the majority of cases, when applying the new rendered machine config, the Operator performs the following steps on each affected node until all of the affected nodes have the updated configuration:

1. Cordon. The MCO marks the node as not schedulable for additional workloads.

2. Drain. The MCO terminates all running workloads on the node, causing the workloads to be rescheduled onto other nodes.

3. Apply. The MCO writes the new configuration to the nodes as needed.

4. Reboot. The MCO restarts the node.

5. Uncordon. The MCO marks the node as schedulable for workloads.

Throughout this process, the MCO maintains the required number of pods based on the **MaxUnavailable** value set in the machine config pool.

If the MCO drains pods on the master node, note the following conditions:

- In single-node OpenShift clusters, the MCO skips the drain operation.

- The MCO does not drain static pods in order to prevent interference with services, such as etcd.

> **NOTE**
>
> In certain cases the nodes are not drained. For more information, see "About the Machine Config Operator."

There are ways to mitigate the disruption caused by drain and reboot cycles by using node disruption policies or disabling control plane reboots. For more information, see "Understanding node restart behaviors after machine config changes" and "Disabling the Machine Config Operator from automatically rebooting."

Additional resources

- About the Machine Config Operator

- [Using node disruption policies to minimize disruption from machine config changes](#)

- [Disabling the Machine Config Operator from automatically rebooting](#)

1.4. UNDERSTANDING CONFIGURATION DRIFT DETECTION

There might be situations when the on-disk state of a node differs from what is configured in the machine config. This is known as *configuration drift*. For example, a cluster admin might manually modify a file, a systemd unit file, or a file permission that was configured through a machine config. This causes configuration drift. Configuration drift can cause problems between nodes in a Machine Config Pool or when the machine configs are updated.

The Machine Config Operator (MCO) uses the Machine Config Daemon (MCD) to check nodes for configuration drift on a regular basis. If detected, the MCO sets the node and the machine config pool (MCP) to **Degraded** and reports the error. A degraded node is online and operational, but, it cannot be updated.

The MCD performs configuration drift detection upon each of the following conditions:

- When a node boots.

- After any of the files (Ignition files and systemd drop-in units) specified in the machine config are modified outside of the machine config.

- Before a new machine config is applied.

> **NOTE**
>
> If you apply a new machine config to the nodes, the MCD temporarily shuts down configuration drift detection. This shutdown is needed because the new machine config necessarily differs from the machine config on the nodes. After the new machine config is applied, the MCD restarts detecting configuration drift using the new machine config.

When performing configuration drift detection, the MCD validates that the file contents and permissions fully match what the currently-applied machine config specifies. Typically, the MCD detects configuration drift in less than a second after the detection is triggered.

If the MCD detects configuration drift, the MCD performs the following tasks:

- Emits an error to the console logs

- Emits a Kubernetes event

- Stops further detection on the node

- Sets the node and MCP to **degraded**

You can check if you have a degraded node by listing the MCPs:

```
$ oc get mcp worker
```

If you have a degraded MCP, the **DEGRADEDMACHINECOUNT** field is non-zero, similar to the following output:

Example output

```
NAME    CONFIG                                  UPDATED  UPDATING  DEGRADED
MACHINECOUNT  READYMACHINECOUNT  UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT  AGE
worker  rendered-worker-404caf3180818d8ac1f50c32f14b57c3  False    True      True      2
1              1                  1                        5h51m
```

You can determine if the problem is caused by configuration drift by examining the machine config pool:

```
$ oc describe mcp worker
```

Example output

```
...
    Last Transition Time:  2021-12-20T18:54:00Z
    Message:                Node ci-ln-j4h8nkb-72292-pxqxz-worker-a-fjks4 is reporting: "content mismatch
for file \"/etc/mco-test-file\"" 1
    Reason:               1 nodes are reporting degraded status on sync
    Status:               True
    Type:                 NodeDegraded 2
...
```

1 This message shows that a node's **/etc/mco-test-file** file, which was added by the machine config, has changed outside of the machine config.

2 The state of the node is **NodeDegraded**.

Or, if you know which node is degraded, examine that node:

```
$ oc describe node/ci-ln-j4h8nkb-72292-pxqxz-worker-a-fjks4
```

Example output

```
...

Annotations:        cloud.network.openshift.io/egress-ipconfig: [{"interface":"nic0","ifaddr":
{"ipv4":"10.0.128.0/17"},"capacity":{"ip":10}}]
                    csi.volume.kubernetes.io/nodeid:
                     {"pd.csi.storage.gke.io":"projects/openshift-gce-devel-ci/zones/us-central1-
a/instances/ci-ln-j4h8nkb-72292-pxqxz-worker-a-fjks4"}
                    machine.openshift.io/machine: openshift-machine-api/ci-ln-j4h8nkb-72292-pxqxz-worker-
a-fjks4
                    machineconfiguration.openshift.io/controlPlaneTopology: HighlyAvailable
                    machineconfiguration.openshift.io/currentConfig: rendered-worker-
67bd55d0b02b0f659aef33680693a9f9
                    machineconfiguration.openshift.io/desiredConfig: rendered-worker-
67bd55d0b02b0f659aef33680693a9f9
                    machineconfiguration.openshift.io/reason: content mismatch for file "/etc/mco-test-file"
1
                    machineconfiguration.openshift.io/state: Degraded 2
...
```

1 The error message indicating that configuration drift was detected between the node and the listed machine config. Here the error message indicates that the contents of the **/etc/mco-test-file**, which was added by the machine config, has changed outside of the machine config.

2 The state of the node is **Degraded**.

You can correct configuration drift and return the node to the **Ready** state by performing one of the following remediations:

- Ensure that the contents and file permissions of the files on the node match what is configured in the machine config. You can manually rewrite the file contents or change the file permissions.

- Generate a force file on the degraded node. The force file causes the MCD to bypass the usual configuration drift detection and reapplies the current machine config.

> **NOTE**
>
> Generating a force file on a node causes that node to reboot.

1.5. CHECKING MACHINE CONFIG POOL STATUS

To see the status of the Machine Config Operator (MCO), its sub-components, and the resources it manages, use the following **oc** commands:

Procedure

1. To see the number of MCO-managed nodes available on your cluster for each machine config pool (MCP), run the following command:

   ```
   $ oc get machineconfigpool
   ```

   Example output

   ```
   NAME    CONFIG              UPDATED UPDATING  DEGRADED MACHINECOUNT
   READYMACHINECOUNT  UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
   AGE
   master   rendered-master-06c9c4…  True   False    False   3       3         3
   0          4h42m
   worker   rendered-worker-f4b64…  False  True     False   3       2         2
   0          4h42m
   ```

   where:

   UPDATED

   The **True** status indicates that the MCO has applied the current machine config to the nodes in that MCP. The current machine config is specified in the **STATUS** field in the **oc get mcp** output. The **False** status indicates a node in the MCP is updating.

   UPDATING

   The **True** status indicates that the MCO is applying the desired machine config, as specified in the **MachineConfigPool** custom resource, to at least one of the nodes in that MCP. The desired machine config is the new, edited machine config. Nodes that are updating might

not be available for scheduling. The **False** status indicates that all nodes in the MCP are updated.

DEGRADED

A **True** status indicates the MCO is blocked from applying the current or desired machine config to at least one of the nodes in that MCP, or the configuration is failing. Nodes that are degraded might not be available for scheduling. A **False** status indicates that all nodes in the MCP are ready.

MACHINECOUNT

Indicates the total number of machines in that MCP.

READYMACHINECOUNT

Indicates the total number of machines in that MCP that are ready for scheduling.

UPDATEDMACHINECOUNT

Indicates the total number of machines in that MCP that have the current machine config.

DEGRADEDMACHINECOUNT

Indicates the total number of machines in that MCP that are marked as degraded or unreconcilable.

In the previous output, there are three control plane (master) nodes and three worker nodes. The control plane MCP and the associated nodes are updated to the current machine config. The nodes in the worker MCP are being updated to the desired machine config. Two of the nodes in the worker MCP are updated and one is still updating, as indicated by the **UPDATEDMACHINECOUNT** being **2**. There are no issues, as indicated by the **DEGRADEDMACHINECOUNT** being **0** and **DEGRADED** being **False**.

While the nodes in the MCP are updating, the machine config listed under **CONFIG** is the current machine config, which the MCP is being updated from. When the update is complete, the listed machine config is the desired machine config, which the MCP was updated to.

> **NOTE**
>
> If a node is being cordoned, that node is not included in the **READYMACHINECOUNT**, but is included in the **MACHINECOUNT**. Also, the MCP status is set to **UPDATING**. Because the node has the current machine config, it is counted in the **UPDATEDMACHINECOUNT** total:
>
> **Example output**
>
> ```
> NAME     CONFIG               UPDATED UPDATING  DEGRADED
> MACHINECOUNT  READYMACHINECOUNT  UPDATEDMACHINECOUNT
> DEGRADEDMACHINECOUNT  AGE
> master   rendered-master-06c9c4… True    False     False    3        3
> 3            0              4h42m
> worker   rendered-worker-c1b41a… False   True      False    3        2
> 3            0              4h42m
> ```

2. To check the status of the nodes in an MCP by examining the **MachineConfigPool** custom resource, run the following command: :

   ```
   $ oc describe mcp worker
   ```

Example output

```
...
  Degraded Machine Count:     0
  Machine Count:              3
  Observed Generation:        2
  Ready Machine Count:        3
  Unavailable Machine Count:  0
  Updated Machine Count:      3
Events:                       <none>
```

> **NOTE**
>
> If a node is being cordoned, the node is not included in the **Ready Machine Count**. It is included in the **Unavailable Machine Count**:
>
> **Example output**
>
> ```
> ...
>   Degraded Machine Count:     0
>   Machine Count:              3
>   Observed Generation:        2
>   Ready Machine Count:        2
>   Unavailable Machine Count:  1
>   Updated Machine Count:      3
> ```

3. To see each existing **MachineConfig** object, run the following command:

   ```
   $ oc get machineconfigs
   ```

   Example output

   ```
   NAME                        GENERATEDBYCONTROLLER         IGNITIONVERSION  AGE
   00-master                   2c9371fbb673b97a6fe8b1c52...  3.2.0            5h18m
   00-worker                   2c9371fbb673b97a6fe8b1c52...  3.2.0            5h18m
   01-master-container-runtime 2c9371fbb673b97a6fe8b1c52...  3.2.0            5h18m
   01-master-kubelet           2c9371fbb673b97a6fe8b1c52…    3.2.0            5h18m
   ...
   rendered-master-dde...        2c9371fbb673b97a6fe8b1c52...  3.2.0          5h18m
   rendered-worker-fde...        2c9371fbb673b97a6fe8b1c52...  3.2.0          5h18m
   ```

   Note that the **MachineConfig** objects listed as **rendered** are not meant to be changed or deleted.

4. To view the contents of a particular machine config (in this case, **01-master-kubelet**), run the following command:

   ```
   $ oc describe machineconfigs 01-master-kubelet
   ```

   The output from the command shows that this **MachineConfig** object contains both configuration files (**cloud.conf** and **kubelet.conf**) and a systemd service (Kubernetes Kubelet):

   Example output

```
Name:        01-master-kubelet
...
Spec:
 Config:
   Ignition:
     Version: 3.2.0
   Storage:
    Files:
      Contents:
        Source:  data:,
      Mode:      420
      Overwrite:  true
      Path:       /etc/kubernetes/cloud.conf
      Contents:
        Source:
data:,kind%3A%20KubeletConfiguration%0AapiVersion%3A%20kubelet.config.k8s.io%2Fv1bet
a1%0Aauthentication%3A%0A%20%20x509%3A%0A%20%20%20%20clientCAFile%3A%20
%2Fetc%2Fkubernetes%2Fkubelet-ca.crt%0A%20%20anonymous...
      Mode:      420
      Overwrite:  true
      Path:       /etc/kubernetes/kubelet.conf
   Systemd:
    Units:
      Contents:  [Unit]
Description=Kubernetes Kubelet
Wants=rpc-statd.service network-online.target crio.service
After=network-online.target crio.service

ExecStart=/usr/bin/hyperkube \
    kubelet \
      --config=/etc/kubernetes/kubelet.conf \ ...
```

If something goes wrong with a machine config that you apply, you can always back out that change. For example, if you had run **oc create -f ./myconfig.yaml** to apply a machine config, you could remove that machine config by running the following command:

```
$ oc delete -f ./myconfig.yaml
```

If that was the only problem, the nodes in the affected pool should return to a non-degraded state. This actually causes the rendered configuration to roll back to its previously rendered state.

If you add your own machine configs to your cluster, you can use the commands shown in the previous example to check their status and the related status of the pool to which they are applied.

1.6. CHECKING MACHINE CONFIG NODE STATUS

During updates you might want to monitor the progress of individual nodes in case issues arise and you need to troubleshoot a node.

To see the status of the Machine Config Operator (MCO) updates to your cluster, use the following **oc** commands:

IMPORTANT

Improved MCO state reporting is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Procedure

1. Get a summary of update statuses for all nodes in all machine config pools by running the following command:

   ```
   $ oc get machineconfignodes
   ```

Example output

```
NAME                     UPDATED  UPDATEPREPARED  UPDATEEXECUTED  UPDATEPOSTACTIONCOMPLETED  UPDATECOMPLETED  RESUMED
ip-10-0-12-194.ec2.internal  True     False           False           False                     False            False
ip-10-0-17-102.ec2.internal  False    True            False           False                     False            False
ip-10-0-2-232.ec2.internal   False    False           True            False                     False            False
ip-10-0-59-251.ec2.internal  False    False           False           True                      False            False
ip-10-0-59-56.ec2.internal   False    False           False           False                     True             True
ip-10-0-6-214.ec2.internal   False    False           Unknown         False                     False            False
```

where:

UPDATED

The **True** status indicates that the MCO has applied the current machine config to that particular node. The **False** status indicates that the node is currently updating. The **Unknown** status means the operation is processing.

UPDATEPREPARED

The **False** status indicates that the MCO has not started reconciling the new machine configs to be distributed. The **True** status indicates that the MCO has completed this phase of the update. The **Unknown** status means the operation is processing.

UPDATEEXECUTED

The **False** status indicates that the MCO has not started cordoning and draining the node. It also indicates that the disk state and operating system have not started updating. The **True** status indicates that the MCO has completed this phase of the update. The **Unknown** status means the operation is processing.

UPDATEPOSTACTIONCOMPLETED

The **False** status indicates that the MCO has not started rebooting the node or closing the daemon. The **True** status indicates that the MCO has completed reboot and updating the

node status. The **Unknown** status indicates either that an error has occurred during the update process at this phase, or that the MCO is currently applying the update.

UPDATECOMPLETED

The **False** status indicates that the MCO has not started uncordoning the node and updating the node state and metrics. The **True** status indicates that the MCO has finished updating the node state and available metrics.

RESUMED

The **False** status indicates that the MCO has not started the config drift monitor. The **True** status indicates that the node has resumed operation. The **Unknown** status means the operation is processing.

> **NOTE**
>
> Within the primary phases previously described, you can have secondary phases which you can use to see the update progression in more detail. You can get more information that includes secondary phases of updates by using the **-o wide** option of the preceding command. This provides the additional **UPDATECOMPATIBLE**, **UPDATEFILESANDOS**, **DRAINEDNODE**, **CORDONEDNODE**, **REBOOTNODE**, **RELOADEDCRIO** and **UNCORDONED** columns. These secondary phases do not always occur and depend on the type of update you want to apply.

2. Check the update status of nodes in a specific machine config pool by running the following command:

```
$ oc get machineconfignodes $(oc get machineconfignodes -o json | jq -r
'.items[]|select(.spec.pool.name=="<pool_name>")|.metadata.name') 1
```

1 The name of the pool is the **MachineConfigPool** object name.

Example output

```
NAME                     UPDATED   UPDATEPREPARED   UPDATEEXECUTED
UPDATEPOSTACTIONCOMPLETE   UPDATECOMPLETE   RESUMED
ip-10-0-48-226.ec2.internal  True      False            False           False                False
False
ip-10-0-5-241.ec2.internal   True      False            False           False                False
False
ip-10-0-74-108.ec2.internal  True      False            False           False                False
False
```

3. Check the update status of an individual node by running the following command:

```
$ oc describe machineconfignode/<node_name> 1
```

1 The name of the node is the **MachineConfigNode** object name.

Example output

```
Name:          <node_name>
Namespace:
Labels:        <none>
Annotations: <none>
API Version:  machineconfiguration.openshift.io/v1alpha1
Kind:          MachineConfigNode
Metadata:
  Creation Timestamp:  2023-10-17T13:08:58Z
  Generation:          1
  Resource Version:    49443
  UID:                 4bd758ab-2187-413c-ac42-882e61761b1d
Spec:
  Node Ref:
    Name:          <node_name>
  Pool:
    Name:        master
  ConfigVersion:
    Desired: rendered-worker-823ff8dc2b33bf444709ed7cd2b9855b
Status:
  Conditions:
    Last Transition Time:  2023-10-17T13:09:02Z
    Message:               Node has completed update to config rendered-master-
cf99e619747ab19165f11e3546c71f1e
    Reason:               NodeUpgradeComplete
    Status:               True
    Type:                 Updated
    Last Transition Time:  2023-10-17T13:09:02Z
    Message:               This node has not yet entered the UpdatePreparing phase
    Reason:               NotYetOccured
    Status:               False
  Config Version:
    Current:              rendered-worker-823ff8dc2b33bf444709ed7cd2b9855b
    Desired:              rendered-worker-823ff8dc2b33bf444709ed7cd2b9855b
  Health:                Healthy
  Most Recent Error:
  Observed Generation: 3
```

1 The desired configuration specified in the **spec.configversion.desired** field updates immediately when a new configuration is detected on the node.

2 The desired configuration specified in the **status.configversion.desired** field updates only when the new configuration is validated by the Machine Config Daemon (MCD). The MCD performs validation by checking the current phase of the update. If the update successfully passes the **UPDATEPREPARED** phase, then the status adds the new configuration.

1.7. UNDERSTANDING MACHINE CONFIG OPERATOR CERTIFICATES

Machine Config Operator certificates are used to secure connections between the Red Hat Enterprise Linux CoreOS (RHCOS) nodes and the Machine Config Server. For more information, see Machine Config Operator certificates.

1.7.1. Viewing and interacting with certificates

The following certificates are handled in the cluster by the Machine Config Controller (MCC) and can be found in the **ControllerConfig** resource:

- **/etc/kubernetes/kubelet-ca.crt**

- **/etc/kubernetes/static-pod-resources/configmaps/cloud-config/ca-bundle.pem**

- **/etc/pki/ca-trust/source/anchors/openshift-config-user-ca-bundle.crt**

The MCC also handles the image registry certificates and its associated user bundle certificate.

You can get information about the listed certificates, including the underyling bundle the certificate comes from, and the signing and subject data.

Procedure

- Get detailed certificate information by running the following command:

  ```
  $ oc get controllerconfig/machine-config-controller -o yaml | yq -y
  '.status.controllerCertificates'
  ```

  ### Example output

  ```
  "controllerCertificates": [
              {
                  "bundleFile": "KubeAPIServerServingCAData",
                  "signer": "<signer_data1>",
                  "subject": "CN=openshift-kube-apiserver-operator_node-system-admin-
  signer@168909215"
              },
              {
                  "bundleFile": "RootCAData",
                  "signer": "<signer_data2>",
                  "subject": "CN=root-ca,OU=openshift"
              }
          ]
  ```

- Get a simpler version of the information found in the ControllerConfig by checking the machine config pool status using the following command:

  ```
  $ oc get mcp master -o yaml | yq -y '.status.certExpirys'
  ```

  ### Example output

  ```
  status:
   certExpirys:
    - bundle: KubeAPIServerServingCAData
      subject: CN=admin-kubeconfig-signer,OU=openshift
    - bundle: KubeAPIServerServingCAData
      subject: CN=kube-csr-signer_@1689585558
    - bundle: KubeAPIServerServingCAData
      subject: CN=kubelet-signer,OU=openshift
    - bundle: KubeAPIServerServingCAData
  ```

```
    subject: CN=kube-apiserver-to-kubelet-signer,OU=openshift
 - bundle: KubeAPIServerServingCAData
   subject: CN=kube-control-plane-signer,OU=openshift
```

This method is meant for OpenShift Container Platform applications that already consume machine config pool information.

- Check which image registry certificates are on the nodes by looking at the contents of the **/etc/docker/cert.d** directory:

```
# ls /etc/docker/certs.d
```

Example output

```
image-registry.openshift-image-registry.svc.cluster.local:5000 image-registry.openshift-
image-registry.svc:5000
```

CHAPTER 2. USING MACHINE CONFIG OBJECTS TO CONFIGURE NODES

You can use the tasks in this section to create **MachineConfig** objects that modify files, systemd unit files, and other operating system features running on OpenShift Container Platform nodes. For more ideas on working with machine configs, see content related to updating SSH authorized keys, verifying image signatures, enabling SCTP, and configuring iSCSI initiatornames for OpenShift Container Platform.

OpenShift Container Platform supports Ignition specification version 3.2. All new machine configs you create going forward should be based on Ignition specification version 3.2. If you are upgrading your OpenShift Container Platform cluster, any existing Ignition specification version 2.x machine configs will be translated automatically to specification version 3.2.

There might be situations where the configuration on a node does not fully match what the currently-applied machine config specifies. This state is called *configuration drift*. The Machine Config Daemon (MCD) regularly checks the nodes for configuration drift. If the MCD detects configuration drift, the MCO marks the node **degraded** until an administrator corrects the node configuration. A degraded node is online and operational, but, it cannot be updated. For more information on configuration drift, see Understanding configuration drift detection .

TIP

Use the following "Configuring chrony time service" procedure as a model for how to go about adding other configuration files to OpenShift Container Platform nodes.

2.1. 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: 4.16.0
   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 ❹
    driftfile /var/lib/chrony/drift
    makestep 1.0 3
    rtcsync
    logdir /var/log/chrony
```

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

❸ 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**.

❹ 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
     ```

Additional resources

- [Creating machine configs with Butane](#)

2.2. DISABLING THE CHRONY TIME SERVICE

You can disable the chrony time service (**chronyd**) for nodes with a specific role by using a **MachineConfig** custom resource (CR).

Prerequisites

- Install the OpenShift CLI (**oc**).

- Log in as a user with **cluster-admin** privileges.

Procedure

1. Create the **MachineConfig** CR that disables **chronyd** for the specified node role.

   a. Save the following YAML in the **disable-chronyd.yaml** file:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: <node_role>    1
  name: disable-chronyd
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - contents: |
            [Unit]
            Description=NTP client/server
            Documentation=man:chronyd(8) man:chrony.conf(5)
            After=ntpdate.service sntp.service ntpd.service
            Conflicts=ntpd.service systemd-timesyncd.service
            ConditionCapability=CAP_SYS_TIME
            [Service]
            Type=forking
            PIDFile=/run/chrony/chronyd.pid
            EnvironmentFile=-/etc/sysconfig/chronyd
            ExecStart=/usr/sbin/chronyd $OPTIONS
            ExecStartPost=/usr/libexec/chrony-helper update-daemon
            PrivateTmp=yes
            ProtectHome=yes
            ProtectSystem=full
            [Install]
            WantedBy=multi-user.target
          enabled: false
          name: "chronyd.service"
```

1 Node role where you want to disable **chronyd**, for example, **master**.

b. Create the **MachineConfig** CR by running the following command:

```
$ oc create -f disable-chronyd.yaml
```

2.3. ADDING KERNEL ARGUMENTS TO NODES

In some special cases, you might want to add kernel arguments to a set of nodes in your cluster. This should only be done with caution and clear understanding of the implications of the arguments you set.

> **WARNING**
>
> Improper use of kernel arguments can result in your systems becoming unbootable.

Examples of kernel arguments you could set include:

- **nosmt**: Disables symmetric multithreading (SMT) in the kernel. Multithreading allows multiple logical threads for each CPU. You could consider **nosmt** in multi-tenant environments to reduce risks from potential cross-thread attacks. By disabling SMT, you essentially choose security over performance.

- **systemd.unified_cgroup_hierarchy**: Enables Linux control group version 2 (cgroup v2). cgroup v2 is the next version of the kernel control group and offers multiple improvements.

> **IMPORTANT**
>
> cgroup v1 is a deprecated feature. Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.
>
> For the most recent list of major functionality that has been deprecated or removed within OpenShift Container Platform, refer to the *Deprecated and removed features* section of the OpenShift Container Platform release notes.

- **enforcing=0**: Configures Security Enhanced Linux (SELinux) to run in permissive mode. In permissive mode, the system acts as if SELinux is enforcing the loaded security policy, including labeling objects and emitting access denial entries in the logs, but it does not actually deny any operations. While not supported for production systems, permissive mode can be helpful for debugging.

> **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.

See Kernel.org kernel parameters for a list and descriptions of kernel arguments.

In the following procedure, you create a **MachineConfig** object that identifies:

- A set of machines to which you want to add the kernel argument. In this case, machines with a worker role.

- Kernel arguments that are appended to the end of the existing kernel arguments.

- A label that indicates where in the list of machine configs the change is applied.

Prerequisites

- Have administrative privilege to a working OpenShift Container Platform cluster.

Procedure

1. List existing **MachineConfig** objects for your OpenShift Container Platform cluster to determine how to label your machine config:

   ```
   $ oc get MachineConfig
   ```

   Example output

   ```
   NAME                                GENERATEDBYCONTROLLER
   IGNITIONVERSION   AGE
   00-master                           52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
   33m
   00-worker                           52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
   33m
   01-master-container-runtime                52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
   3.2.0         33m
   01-master-kubelet                   52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
   3.2.0         33m
   01-worker-container-runtime                52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
   3.2.0         33m
   01-worker-kubelet                   52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
   3.2.0         33m
   99-master-generated-registries             52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
   3.2.0         33m
   99-master-ssh                                               3.2.0         40m
   99-worker-generated-registries             52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
   3.2.0         33m
   99-worker-ssh                                               3.2.0         40m
   rendered-master-23e785de7587df95a4b517e0647e5ab7
   52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0         33m
   rendered-worker-5d596d9293ca3ea80c896a1191735bb1
   52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0         33m
   ```

2. Create a **MachineConfig** object file that identifies the kernel argument (for example, **05-worker-kernelarg-selinuxpermissive.yaml**)

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: worker ❶
     name: 05-worker-kernelarg-selinuxpermissive ❷
   spec:
     kernelArguments:
       - enforcing=0 ❸
   ```

   ❶ Applies the new kernel argument only to worker nodes.

   ❷ Named to identify where it fits among the machine configs (05) and what it does (adds a kernel argument to configure SELinux permissive mode).

   ❸ Identifies the exact kernel argument as **enforcing=0**.

3. Create the new machine config:

```
$ oc create -f 05-worker-kernelarg-selinuxpermissive.yaml
```

4. Check the machine configs to see that the new one was added:

```
$ oc get MachineConfig
```

Example output

```
NAME                                     GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
00-worker                                52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
01-master-container-runtime                     52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0         33m
01-master-kubelet                        52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0         33m
01-worker-container-runtime                     52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0         33m
01-worker-kubelet                        52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0         33m
05-worker-kernelarg-selinuxpermissive                               3.2.0         105s
99-master-generated-registries                   52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0         33m
99-master-ssh                                                       3.2.0         40m
99-worker-generated-registries                   52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0         33m
99-worker-ssh                                                       3.2.0         40m
rendered-master-23e785de7587df95a4b517e0647e5ab7
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0         33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0         33m
```

5. Check the nodes:

```
$ oc get nodes
```

Example output

```
NAME                        STATUS                ROLES   AGE  VERSION
ip-10-0-136-161.ec2.internal  Ready                 worker  28m  v1.29.4
ip-10-0-136-243.ec2.internal  Ready                 master  34m  v1.29.4
ip-10-0-141-105.ec2.internal  Ready,SchedulingDisabled  worker  28m  v1.29.4
ip-10-0-142-249.ec2.internal  Ready                 master  34m  v1.29.4
ip-10-0-153-11.ec2.internal   Ready                 worker  28m  v1.29.4
ip-10-0-153-150.ec2.internal  Ready                 master  34m  v1.29.4
```

You can see that scheduling on each worker node is disabled as the change is being applied.

6. Check that the kernel argument worked by going to one of the worker nodes and listing the kernel command line arguments (in **/proc/cmdline** on the host):

```
$ oc debug node/ip-10-0-141-105.ec2.internal
```

Example output

```
Starting pod/ip-10-0-141-105ec2internal-debug ...
To use host binaries, run `chroot /host`

sh-4.2# cat /host/proc/cmdline
BOOT_IMAGE=/ostree/rhcos-... console=tty0 console=ttyS0,115200n8
rootflags=defaults,prjquota rw root=UUID=fd0... ostree=/ostree/boot.0/rhcos/16...
coreos.oem.id=qemu coreos.oem.id=ec2 ignition.platform.id=ec2 enforcing=0

sh-4.2# exit
```

You should see the **enforcing=0** argument added to the other kernel arguments.

2.4. ENABLING MULTIPATHING WITH KERNEL ARGUMENTS ON RHCOS

IMPORTANT

Enabling multipathing during installation is supported and recommended for nodes provisioned in OpenShift Container Platform 4.8 or later versions. In setups where any I/O to non-optimized paths results in I/O system errors, you must enable multipathing at installation time. For more information about enabling multipathing during installation time, see "Enabling multipathing post installation" in the *Installing on bare metal* documentation.

Red Hat Enterprise Linux CoreOS (RHCOS) supports multipathing on the primary disk, allowing stronger resilience to hardware failure to achieve higher host availability. Postinstallation support is available by activating multipathing via the machine config.

IMPORTANT

On IBM Z® and IBM® LinuxONE, you can enable multipathing only if you configured your cluster for it during installation. For more information, see "Installing RHCOS and starting the OpenShift Container Platform bootstrap process" in *Installing a cluster with z/VM on IBM Z® and IBM® LinuxONE*.

IMPORTANT

When a OpenShift Container Platform 4.16 cluster is installed or configured as a post-installation activity on a single VIOS host with "vSCSI" storage on IBM Power® with multipath configured, the CoreOS nodes with multipath enabled fail to boot. This behavior is expected, as only one path is available to the Node.

Prerequisites

- You have a running OpenShift Container Platform cluster that uses version 4.7 or later.

- You are logged in to the cluster as a user with administrative privileges.

- You have confirmed that the disk is enabled for multipathing. Multipathing is only supported on hosts that are connected to a SAN via an HBA adapter.

Procedure

1. To enable multipathing postinstallation on control plane nodes:

   - Create a machine config file, such as **99-master-kargs-mpath.yaml**, that instructs the cluster to add the **master** label and that identifies the multipath kernel argument, for example:

     ```
     apiVersion: machineconfiguration.openshift.io/v1
     kind: MachineConfig
     metadata:
       labels:
         machineconfiguration.openshift.io/role: "master"
       name: 99-master-kargs-mpath
     spec:
       kernelArguments:
         - 'rd.multipath=default'
         - 'root=/dev/disk/by-label/dm-mpath-root'
     ```

2. To enable multipathing postinstallation on worker nodes:

   - Create a machine config file, such as **99-worker-kargs-mpath.yaml**, that instructs the cluster to add the **worker** label and that identifies the multipath kernel argument, for example:

     ```
     apiVersion: machineconfiguration.openshift.io/v1
     kind: MachineConfig
     metadata:
       labels:
         machineconfiguration.openshift.io/role: "worker"
       name: 99-worker-kargs-mpath
     spec:
       kernelArguments:
         - 'rd.multipath=default'
         - 'root=/dev/disk/by-label/dm-mpath-root'
     ```

3. Create the new machine config by using either the master or worker YAML file you previously created:

   ```
   $ oc create -f ./99-worker-kargs-mpath.yaml
   ```

4. Check the machine configs to see that the new one was added:

   ```
   $ oc get MachineConfig
   ```

   Example output

   ```
   NAME                           GENERATEDBYCONTROLLER
   IGNITIONVERSION   AGE
   00-master                      52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
   33m
   00-worker                      52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
   33m
   01-master-container-runtime            52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
   3.2.0            33m
   ```

```
01-master-kubelet                        52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
01-worker-container-runtime               52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
01-worker-kubelet                        52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
99-master-generated-registries            52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
99-master-ssh                                           3.2.0          40m
99-worker-generated-registries            52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          33m
99-worker-kargs-mpath                    52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0          105s
99-worker-ssh                                           3.2.0          40m
rendered-master-23e785de7587df95a4b517e0647e5ab7
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0          33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0          33m
```

5. Check the nodes:

```
$ oc get nodes
```

Example output

```
NAME                      STATUS              ROLES    AGE   VERSION
ip-10-0-136-161.ec2.internal   Ready                   worker   28m   v1.29.4
ip-10-0-136-243.ec2.internal   Ready                   master   34m   v1.29.4
ip-10-0-141-105.ec2.internal   Ready,SchedulingDisabled   worker   28m   v1.29.4
ip-10-0-142-249.ec2.internal   Ready                   master   34m   v1.29.4
ip-10-0-153-11.ec2.internal    Ready                   worker   28m   v1.29.4
ip-10-0-153-150.ec2.internal   Ready                   master   34m   v1.29.4
```

You can see that scheduling on each worker node is disabled as the change is being applied.

6. Check that the kernel argument worked by going to one of the worker nodes and listing the kernel command line arguments (in **/proc/cmdline** on the host):

```
$ oc debug node/ip-10-0-141-105.ec2.internal
```

Example output

```
Starting pod/ip-10-0-141-105ec2internal-debug ...
To use host binaries, run `chroot /host`

sh-4.2# cat /host/proc/cmdline
...
rd.multipath=default root=/dev/disk/by-label/dm-mpath-root
...

sh-4.2# exit
```

You should see the added kernel arguments.

Additional resources

- See Enabling multipathing with kernel arguments on RHCOS for more information about enabling multipathing during installation time.

2.5. ADDING A REAL-TIME KERNEL TO NODES

Some OpenShift Container Platform workloads require a high degree of determinism.While Linux is not a real-time operating system, the Linux real-time kernel includes a preemptive scheduler that provides the operating system with real-time characteristics.

If your OpenShift Container Platform workloads require these real-time characteristics, you can switch your machines to the Linux real-time kernel. For OpenShift Container Platform, 4.16 you can make this switch using a **MachineConfig** object. Although making the change is as simple as changing a machine config **kernelType** setting to **realtime**, there are a few other considerations before making the change:

- Currently, real-time kernel is supported only on worker nodes, and only for radio access network (RAN) use.

- The following procedure is fully supported with bare metal installations that use systems that are certified for Red Hat Enterprise Linux for Real Time 8.

- Real-time support in OpenShift Container Platform is limited to specific subscriptions.

- The following procedure is also supported for use with Google Cloud Platform.

Prerequisites

- Have a running OpenShift Container Platform cluster (version 4.4 or later).

- Log in to the cluster as a user with administrative privileges.

Procedure

1. Create a machine config for the real-time kernel: Create a YAML file (for example, **99-worker-realtime.yaml**) that contains a **MachineConfig** object for the **realtime** kernel type. This example tells the cluster to use a real-time kernel for all worker nodes:

   ```
   $ cat << EOF > 99-worker-realtime.yaml
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: "worker"
     name: 99-worker-realtime
   spec:
     kernelType: realtime
   EOF
   ```

2. Add the machine config to the cluster. Type the following to add the machine config to the cluster:

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

3. Check the real-time kernel: Once each impacted node reboots, log in to the cluster and run the following commands to make sure that the real-time kernel has replaced the regular kernel for the set of nodes you configured:

```
$ oc get nodes
```

Example output

```
NAME                              STATUS  ROLES    AGE   VERSION
ip-10-0-143-147.us-east-2.compute.internal  Ready   worker   103m  v1.29.4
ip-10-0-146-92.us-east-2.compute.internal   Ready   worker   101m  v1.29.4
ip-10-0-169-2.us-east-2.compute.internal    Ready   worker   102m  v1.29.4
```

```
$ oc debug node/ip-10-0-143-147.us-east-2.compute.internal
```

Example output

```
Starting pod/ip-10-0-143-147us-east-2computeinternal-debug ...
To use host binaries, run `chroot /host`

sh-4.4# uname -a
Linux <worker_node> 4.18.0-147.3.1.rt24.96.el8_1.x86_64 #1 SMP PREEMPT RT
    Wed Nov 27 18:29:55 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
```

The kernel name contains **rt** and text "PREEMPT RT" indicates that this is a real-time kernel.

4. To go back to the regular kernel, delete the **MachineConfig** object:

```
$ oc delete -f 99-worker-realtime.yaml
```

2.6. CONFIGURING JOURNALD SETTINGS

If you need to configure settings for the **journald** service on OpenShift Container Platform nodes, you can do that by modifying the appropriate configuration file and passing the file to the appropriate pool of nodes as a machine config.

This procedure describes how to modify **journald** rate limiting settings in the **/etc/systemd/journald.conf** file and apply them to worker nodes. See the **journald.conf** man page for information on how to use that file.

Prerequisites

- Have a running OpenShift Container Platform cluster.

- Log in to the cluster as a user with administrative privileges.

Procedure

1. Create a Butane config file, **40-worker-custom-journald.bu**, that includes an **/etc/systemd/journald.conf** file with the required settings.

> **NOTE**
>
> See "Creating machine configs with Butane" for information about Butane.

```
variant: openshift
version: 4.16.0
metadata:
  name: 40-worker-custom-journald
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
  - path: /etc/systemd/journald.conf
    mode: 0644
    overwrite: true
    contents:
      inline: |
        # Disable rate limiting
        RateLimitInterval=1s
        RateLimitBurst=10000
        Storage=volatile
        Compress=no
        MaxRetentionSec=30s
```

2. Use Butane to generate a **MachineConfig** object file, **40-worker-custom-journald.yaml**, containing the configuration to be delivered to the worker nodes:

```
$ butane 40-worker-custom-journald.bu -o 40-worker-custom-journald.yaml
```

3. Apply the machine config to the pool:

```
$ oc apply -f 40-worker-custom-journald.yaml
```

4. Check that the new machine config is applied and that the nodes are not in a degraded state. It might take a few minutes. The worker pool will show the updates in progress, as each node successfully has the new machine config applied:

```
$ oc get machineconfigpool
NAME   CONFIG           UPDATED UPDATING DEGRADED MACHINECOUNT
READYMACHINECOUNT UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
AGE
master rendered-master-35 True    False    False    3        3         3          0
34m
worker rendered-worker-d8 False   True     False    3        1         1          0
34m
```

5. To check that the change was applied, you can log in to a worker node:

```
$ oc get node | grep worker
ip-10-0-0-1.us-east-2.compute.internal   Ready   worker   39m   v0.0.0-master+$Format:%h$
$ oc debug node/ip-10-0-0-1.us-east-2.compute.internal
Starting pod/ip-10-0-141-142us-east-2computeinternal-debug ...

...
sh-4.2# chroot /host
```

```
sh-4.4# cat /etc/systemd/journald.conf
# Disable rate limiting
RateLimitInterval=1s
RateLimitBurst=10000
Storage=volatile
Compress=no
MaxRetentionSec=30s
sh-4.4# exit
```

Additional resources

- Creating machine configs with Butane

2.7. ADDING EXTENSIONS TO RHCOS

RHCOS is a minimal container-oriented RHEL operating system, designed to provide a common set of capabilities to OpenShift Container Platform clusters across all platforms. While adding software packages to RHCOS systems is generally discouraged, the MCO provides an **extensions** feature you can use to add a minimal set of features to RHCOS nodes.

Currently, the following extensions are available:

- **usbguard**: Adding the **usbguard** extension protects RHCOS systems from attacks from intrusive USB devices. See USBGuard for details.

- **kerberos**: Adding the **kerberos** extension provides a mechanism that allows both users and machines to identify themselves to the network to receive defined, limited access to the areas and services that an administrator has configured. See Using Kerberos for details, including how to set up a Kerberos client and mount a Kerberized NFS share.

The following procedure describes how to use a machine config to add one or more extensions to your RHCOS nodes.

Prerequisites

- Have a running OpenShift Container Platform cluster (version 4.6 or later).

- Log in to the cluster as a user with administrative privileges.

Procedure

1. Create a machine config for extensions: Create a YAML file (for example, **80-extensions.yaml**) that contains a **MachineConfig extensions** object. This example tells the cluster to add the **usbguard** extension.

   ```
   $ cat << EOF > 80-extensions.yaml
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: worker
     name: 80-worker-extensions
   spec:
     config:
       ignition:
   ```

```
      version: 3.2.0
   extensions:
     - usbguard
   EOF
```

2. Add the machine config to the cluster. Type the following to add the machine config to the cluster:

```
$ oc create -f 80-extensions.yaml
```

This sets all worker nodes to have rpm packages for **usbguard** installed.

3. Check that the extensions were applied:

```
$ oc get machineconfig 80-worker-extensions
```

Example output

```
NAME                GENERATEDBYCONTROLLER IGNITIONVERSION AGE
80-worker-extensions                3.2.0         57s
```

4. Check that the new machine config is now applied and that the nodes are not in a degraded state. It may take a few minutes. The worker pool will show the updates in progress, as each machine successfully has the new machine config applied:

```
$ oc get machineconfigpool
```

Example output

```
NAME   CONFIG         UPDATED UPDATING DEGRADED MACHINECOUNT
READYMACHINECOUNT UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
AGE
master rendered-master-35 True    False    False   3        3          3            0
34m
worker rendered-worker-d8 False   True     False   3        1          1            0
34m
```

5. Check the extensions. To check that the extension was applied, run:

```
$ oc get node | grep worker
```

Example output

```
NAME                         STATUS  ROLES   AGE   VERSION
ip-10-0-169-2.us-east-2.compute.internal   Ready   worker   102m  v1.29.4
```

```
$ oc debug node/ip-10-0-169-2.us-east-2.compute.internal
```

Example output

```
...
```

> To use host binaries, run `chroot /host`
> sh-4.4# chroot /host
> sh-4.4# rpm -q usbguard
> usbguard-0.7.4-4.el8.x86_64.rpm

2.8. LOADING CUSTOM FIRMWARE BLOBS IN THE MACHINE CONFIG MANIFEST

Because the default location for firmware blobs in **/usr/lib** is read-only, you can locate a custom firmware blob by updating the search path. This enables you to load local firmware blobs in the machine config manifest when the blobs are not managed by RHCOS.

Procedure

1. Create a Butane config file, **98-worker-firmware-blob.bu**, that updates the search path so that it is root-owned and writable to local storage. The following example places the custom blob file from your local workstation onto nodes under **/var/lib/firmware**.

   > **NOTE**
   >
   > See "Creating machine configs with Butane" for information about Butane.

   Butane config file for custom firmware blob

   ```
   variant: openshift
   version: 4.16.0
   metadata:
     labels:
       machineconfiguration.openshift.io/role: worker
     name: 98-worker-firmware-blob
   storage:
     files:
     - path: /var/lib/firmware/<package_name>     1
       contents:
         local: <package_name>     2
       mode: 0644     3
   openshift:
     kernel_arguments:
       - 'firmware_class.path=/var/lib/firmware'     4
   ```

   1   Sets the path on the node where the firmware package is copied to.

   2   Specifies a file with contents that are read from a local file directory on the system running Butane. The path of the local file is relative to a **files-dir** directory, which must be specified by using the **--files-dir** option with Butane in the following step.

   3   Sets the permissions for the file on the RHCOS node. It is recommended to set **0644** permissions.

   4   The **firmware_class.path** parameter customizes the kernel search path of where to look for the custom firmware blob that was copied from your local workstation onto the root file system of the node. This example uses **/var/lib/firmware** as the customized path.

2. Run Butane to generate a **MachineConfig** object file that uses a copy of the firmware blob on your local workstation named **98-worker-firmware-blob.yaml**. The firmware blob contains the configuration to be delivered to the nodes. The following example uses the **--files-dir** option to specify the directory on your workstation where the local file or files are located:

```
$ butane 98-worker-firmware-blob.bu -o 98-worker-firmware-blob.yaml --files-dir
<directory_including_package_name>
```

3. Apply the configurations to the nodes 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 98-worker-firmware-blob.yaml
     ```

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

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

Additional resources

- [Creating machine configs with Butane](#)

2.9. CHANGING THE CORE USER PASSWORD FOR NODE ACCESS

By default, Red Hat Enterprise Linux CoreOS (RHCOS) creates a user named **core** on the nodes in your cluster. You can use the **core** user to access the node through a cloud provider serial console or a bare metal baseboard controller manager (BMC). This can be helpful, for example, if a node is down and you cannot access that node by using SSH or the **oc debug node** command. However, by default, there is no password for this user, so you cannot log in without creating one.

You can create a password for the **core** user by using a machine config. The Machine Config Operator (MCO) assigns the password and injects the password into the **/etc/shadow** file, allowing you to log in with the **core** user. The MCO does not examine the password hash. As such, the MCO cannot report if there is a problem with the password.

> **NOTE**
>
> - The password works only through a cloud provider serial console or a BMC. It does not work with SSH.
>
> - If you have a machine config that includes an **/etc/shadow** file or a systemd unit that sets a password, it takes precedence over the password hash.

You can change the password, if needed, by editing the machine config you used to create the password. Also, you can remove the password by deleting the machine config. Deleting the machine config does not remove the user account.

Procedure

1. Using a tool that is supported by your operating system, create a hashed password. For example, create a hashed password using **mkpasswd** by running the following command:

   ```
   $ mkpasswd -m SHA-512 testpass
   ```

   Example output

   ```
   $
   $6$CBZwA6s6AVFOtiZe$aUKDWpthhJEyR3nnhM02NM1sKCpHn9XN.NPrJNQ3HYewioaorp
   wL3mKGLxvW0AOb4pJxqoqP4nFX77y0p00.8.
   ```

2. Create a machine config file that contains the **core** username and the hashed password:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: worker
     name: set-core-user-password
   spec:
     config:
       ignition:
         version: 3.2.0
       passwd:
         users:
         - name: core        1
           passwordHash: <password>    2
   ```

   1 This must be **core**.

   2 The hashed password to use with the **core** account.

3. Create the machine config by running the following command:

   ```
   $ oc create -f <file-name>.yaml
   ```

   The nodes do not reboot and should become available in a few moments. You can use the **oc get mcp** to watch for the machine config pools to be updated, as shown in the following example:

   ```
   NAME     CONFIG                                      UPDATED   UPDATING   DEGRADED
   MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
   DEGRADEDMACHINECOUNT   AGE
   master   rendered-master-d686a3ffc8fdec47280afec446fce8dd   True      False      False      3
   3              3                   0                       64m
   worker   rendered-worker-4605605a5b1f9de1d061e9d350f251e5   False     True       False      3
   3              0                   0                   0                       64m
   ```

Verification

1. After the nodes return to the **UPDATED=True** state, start a debug session for a node by running the following command:

```
$ oc debug node/<node_name>
```

2. Set /**host** as the root directory within the debug shell by running the following command:

```
sh-4.4# chroot /host
```

3. Check the contents of the /**etc**/**shadow** file:

Example output

```
...
core:$6$2sE/010goDuRSxxv$o18K52wor.wlwZp:19418:0:99999:7:::
...
```

The hashed password is assigned to the **core** user.

CHAPTER 3. USING NODE DISRUPTION POLICIES TO MINIMIZE DISRUPTION FROM MACHINE CONFIG CHANGES

By default, when you make certain changes to the fields in a **MachineConfig** object, the Machine Config Operator (MCO) drains and reboots the nodes associated with that machine config. However, you can create a *node disruption policy* that defines a set of changes to some Ignition config objects that would require little or no disruption to your workloads.

A node disruption policy allows you to define the configuration changes that cause a disruption to your cluster, and which changes do not. This allows you to reduce node downtime when making small machine configuration changes in your cluster. To configure the policy, you modify the **MachineConfiguration** object, which is in the **openshift-machine-config-operator** namespace. See the example node disruption policies in the **MachineConfiguration** objects that follow.

> **NOTE**
>
> There are machine configuration changes that always require a reboot, regardless of any node disruption policies. For more information, see *About the Machine Config Operator* .

After you create the node disruption policy, the MCO validates the policy to search for potential issues in the file, such as problems with formatting. The MCO then merges the policy with the cluster defaults and populates the **status.nodeDisruptionPolicyStatus** fields in the machine config with the actions to be performed upon future changes to the machine config. The configurations in your policy always overwrite the cluster defaults.

> **IMPORTANT**
>
> The MCO does not validate whether a change can be successfully applied by your node disruption policy. Therefore, you are responsible to ensure the accuracy of your node disruption policies.

For example, you can configure a node disruption policy so that sudo configurations do not require a node drain and reboot. Or, you can configure your cluster so that updates to **sshd** are applied with only a reload of that one service.

> **IMPORTANT**
>
> The node disruption policy feature is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.
>
> For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

You can control the behavior of the MCO when making the changes to the following Ignition configuration objects:

- **configuration files**: You add to or update the files in the /**var** or /**etc** directory.

- **systemd units**: You create and set the status of a systemd service or modify an existing systemd service.

- **users and groups**: You change SSH keys in the **passwd** section postinstallation.

- **ICSP**, **ITMS**, **IDMS** objects: You can remove mirroring rules from an **ImageContentSourcePolicy** (ICSP), **ImageTagMirrorSet** (ITMS), and **ImageDigestMirrorSet** (IDMS) object.

When you make any of these changes, the node disruption policy determines which of the following actions are required when the MCO implements the changes:

- **Reboot**: The MCO drains and reboots the nodes. This is the default behavior.

- **None**: The MCO does not drain or reboot the nodes. The MCO applies the changes with no further action.

- **Drain**: The MCO cordons and drains the nodes of their workloads. The workloads restart with the new configurations.

- **Reload**: For services, the MCO reloads the specified services without restarting the service.

- **Restart**: For services, the MCO fully restarts the specified services.

- **DaemonReload**: The MCO reloads the systemd manager configuration.

- **Special**: This is an internal MCO-only action and cannot be set by the user.

> **NOTE**
>
> - The **Reboot** and **None** actions cannot be used with any other actions, as the **Reboot** and **None** actions override the others.
>
> - Actions are applied in the order that they are set in the node disruption policy list.
>
> - If you make other machine config changes that do require a reboot or other disruption to the nodes, that reboot supercedes the node disruption policy actions.

3.1. EXAMPLE NODE DISRUPTION POLICIES

The following example **MachineConfiguration** objects contain a node disruption policy.

> **TIP**
>
> A **MachineConfiguration** object and a **MachineConfig** object are different objects. A **MachineConfiguration** object is a singleton object in the MCO namespace that contains configuration parameters for the MCO operator. A **MachineConfig** object defines changes that are applied to a machine config pool.

The following example **MachineConfiguration** object shows no user defined policies. The default node disruption policy values are shown in the **status** stanza.

Default node disruption policy

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
  name: cluster
spec:
  logLevel: Normal
  managementState: Managed
  operatorLogLevel: Normal
status:
  nodeDisruptionPolicyStatus:
    clusterPolicies:
      files:
      - actions:
        - type: None
        path: /etc/mco/internal-registry-pull-secret.json
      - actions:
        - type: None
        path: /var/lib/kubelet/config.json
      - actions:
        - reload:
            serviceName: crio.service
          type: Reload
        path: /etc/machine-config-daemon/no-reboot/containers-gpg.pub
      - actions:
        - reload:
            serviceName: crio.service
          type: Reload
        path: /etc/containers/policy.json
      - actions:
        - type: Special
        path: /etc/containers/registries.conf
      sshkey:
        actions:
        - type: None
  readyReplicas: 0
```

In the following example, when changes are made to the SSH keys, the MCO drains the cluster nodes, reloads the **crio.service**, reloads the systemd configuration, and restarts the **crio-service**.

Example node disruption policy for an SSH key change

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  name: cluster
  namespace: openshift-machine-config-operator
# ...
spec:
  nodeDisruptionPolicy:
    sshkey:
      actions:
      - type: Drain
      - reload:
          serviceName: crio.service
        type: Reload
      - type: DaemonReload
```

```
      - restart:
          serviceName: crio.service
        type: Restart
    # ...
```

In the following example, when changes are made to the files in the **/etc/chrony.conf** directory, the MCO reloads the **chronyd.service** on the cluster nodes.

Example node disruption policy for a configuration file change

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  name: cluster
  namespace: openshift-machine-config-operator
# ...
spec:
  nodeDisruptionPolicy:
    files:
    - actions:
      - reload:
          serviceName: chronyd.service
        type: Reload
      path: /etc/chrony.conf
```

In the following example, when changes are made to the **auditd.service** systemd unit, the MCO drains the cluster nodes, reloads the **crio.service**, reloads the systemd manager configuration, and restarts the **crio.service**.

Example node disruption policy for a configuration file change

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  name: cluster
  namespace: openshift-machine-config-operator
# ...
spec:
  nodeDisruptionPolicy:
    units:
    - name: auditd.service
      actions:
      - type: Drain
      - type: Reload
        reload:
          serviceName: crio.service
      - type: DaemonReload
      - type: Restart
        restart:
          serviceName: crio.service
```

In the following example, when changes are made to the files in the **registries.conf** directory, the MCO does not drain or reboot the nodes and applies the changes with no further action.

Example node disruption policy for a configuration file change

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  name: cluster
  namespace: openshift-machine-config-operator
# ...
spec:
  nodeDisruptionPolicy:
    - actions:
      - type: None
      path: /etc/containers/registries.conf
```

3.2. CONFIGURING NODE RESTART BEHAVIORS UPON MACHINE CONFIG CHANGES

You can create a node disruption policy to define the machine configuration changes that cause a disruption to your cluster, and which changes do not.

You can control how your nodes respond to changes in the files in the **/var** or **/etc** directory, the systemd units, the SSH keys, and the **registries.conf** file.

When you make any of these changes, the node disruption policy determines which of the following actions are required when the MCO implements the changes:

- **Reboot**: The MCO drains and reboots the nodes. This is the default behavior.

- **None**: The MCO does not drain or reboot the nodes. The MCO applies the changes with no further action.

- **Drain**: The MCO cordons and drains the nodes of their workloads. The workloads restart with the new configurations.

- **Reload**: For services, the MCO reloads the specified services without restarting the service.

- **Restart**: For services, the MCO fully restarts the specified services.

- **DaemonReload**: The MCO reloads the systemd manager configuration.

- **Special**: This is an internal MCO-only action and cannot be set by the user.

> **NOTE**
>
> - The **Reboot** and **None** actions cannot be used with any other actions, as the **Reboot** and **None** actions override the others.
>
> - Actions are applied in the order that they are set in the node disruption policy list.
>
> - If you make other machine config changes that do require a reboot or other disruption to the nodes, that reboot supercedes the node disruption policy actions.

Prerequisites

- You have enabled the **TechPreviewNoUpgrade** feature set by using the feature gates. For more information, see "Enabling features using feature gates".

> **WARNING**
>
> Enabling the **TechPreviewNoUpgrade** feature set on your cluster prevents minor version updates. The **TechPreviewNoUpgrade** feature set cannot be disabled. Do not enable this feature set on production clusters.

Procedure

1. Edit the **machineconfigurations.operator.openshift.io** object to define the node disruption policy:

   ```
   $ oc edit MachineConfiguration cluster -n openshift-machine-config-operator
   ```

2. Add a node disruption policy similar to the following:

   ```yaml
   apiVersion: operator.openshift.io/v1
   kind: MachineConfiguration
   metadata:
     name: cluster
   # ...
   spec:
     nodeDisruptionPolicy: 1
       files: 2
       - actions: 3
         - reload: 4
             serviceName: chronyd.service 5
           type: Reload
         path: /etc/chrony.conf 6
       sshkey: 7
         actions:
         - type: Drain
         - reload:
             serviceName: crio.service
           type: Reload
         - type: DaemonReload
         - restart:
             serviceName: crio.service
           type: Restart
       units: 8
       - actions:
         - type: Drain
         - reload:
             serviceName: crio.service
           type: Reload
         - type: DaemonReload
         - restart:
             serviceName: crio.service
           type: Restart
         name: test.service
   ```

1   Specifies the node disruption policy.

2   Specifies a list of machine config file definitions and actions to take to changes on those paths. This list supports a maximum of 50 entries.

3   Specifies the series of actions to be executed upon changes to the specified files. Actions are applied in the order that they are set in this list. This list supports a maximum of 10 entries.

4   Specifies that the listed service is to be reloaded upon changes to the specified files.

5   Specifies the full name of the service to be acted upon.

6   Specifies the location of a file that is managed by a machine config. The actions in the policy apply when changes are made to the file in **path**.

7   Specifies a list of service names and actions to take upon changes to the SSH keys in the cluster.

8   Specifies a list of systemd unit names and actions to take upon changes to those units.

Verification

- View the **MachineConfiguration** object file that you created:

  ```
  $ oc get MachineConfiguration/cluster -o yaml
  ```

  Example output

  ```
  apiVersion: operator.openshift.io/v1
  kind: MachineConfiguration
  metadata:
    labels:
      machineconfiguration.openshift.io/role: worker
    name: cluster
  # ...
  status:
    nodeDisruptionPolicyStatus: 1
      clusterPolicies:
        files:
  # ...
          - actions:
            - reload:
                serviceName: chronyd.service
              type: Reload
            path: /etc/chrony.conf
          sshkey:
            actions:
            - type: Drain
            - reload:
                serviceName: crio.service
              type: Reload
            - type: DaemonReload
            - restart:
                serviceName: crio.service
  ```

```
      type: Restart
    units:
    - actions:
      - type: Drain
      - reload:
          serviceName: crio.service
        type: Reload
      - type: DaemonReload
      - restart:
          serviceName: crio.service
        type: Restart
      name: test.se
# ...
```

1 Specifies the current cluster-validated policies.

CHAPTER 4. CONFIGURING MCO-RELATED CUSTOM RESOURCES

Besides managing **MachineConfig** objects, the MCO manages two custom resources (CRs): **KubeletConfig** and **ContainerRuntimeConfig**. Those CRs let you change node-level settings impacting how the kubelet and CRI-O container runtime services behave.

4.1. CREATING A KUBELETCONFIG CRD TO EDIT KUBELET PARAMETERS

The kubelet configuration is currently serialized as an Ignition configuration, so it can be directly edited. However, there is also a new **kubelet-config-controller** added to the Machine Config Controller (MCC). This lets you use a **KubeletConfig** custom resource (CR) to edit the kubelet parameters.

> **NOTE**
>
> As the fields in the **kubeletConfig** object are passed directly to the kubelet from upstream Kubernetes, the kubelet validates those values directly. Invalid values in the **kubeletConfig** object might cause cluster nodes to become unavailable. For valid values, see the Kubernetes documentation.

Consider the following guidance:

- Edit an existing **KubeletConfig** CR to modify existing settings or add new settings, instead of creating a CR for each change. It is recommended that you create a CR only to modify a different machine config pool, or for changes that are intended to be temporary, so that you can revert the changes.

- Create one **KubeletConfig** CR for each machine config pool with all the config changes you want for that pool.

- As needed, create multiple **KubeletConfig** CRs with a limit of 10 per cluster. For the first **KubeletConfig** CR, the Machine Config Operator (MCO) creates a machine config appended with **kubelet**. With each subsequent CR, the controller creates another **kubelet** machine config with a numeric suffix. For example, if you have a **kubelet** machine config with a **-2** suffix, the next **kubelet** machine config is appended with **-3**.

> **NOTE**
>
> If you are applying a kubelet or container runtime config to a custom machine config pool, the custom role in the **machineConfigSelector** must match the name of the custom machine config pool.
>
> For example, because the following custom machine config pool is named **infra**, the custom role must also be **infra**:
>
> ```
> apiVersion: machineconfiguration.openshift.io/v1
> kind: MachineConfigPool
> metadata:
>   name: infra
> spec:
>   machineConfigSelector:
>     matchExpressions:
>       - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,infra]}
> # ...
> ```

If you want to delete the machine configs, delete them in reverse order to avoid exceeding the limit. For example, you delete the **kubelet-3** machine config before deleting the **kubelet-2** machine config.

> **NOTE**
>
> If you have a machine config with a **kubelet-9** suffix, and you create another **KubeletConfig** CR, a new machine config is not created, even if there are fewer than 10 **kubelet** machine configs.

Example **KubeletConfig** CR

```
$ oc get kubeletconfig
```

```
NAME          AGE
set-max-pods     15m
```

Example showing a **KubeletConfig** machine config

```
$ oc get mc | grep kubelet
```

```
...
99-worker-generated-kubelet-1               b5c5119de007945b6fe6fb215db3b8e2ceb12511   3.2.0
26m
...
```

The following procedure is an example to show how to configure the maximum number of pods per node on the worker nodes.

Prerequisites

1. Obtain the label associated with the static **MachineConfigPool** CR for the type of node you want to configure. Perform one of the following steps:

   a. View the machine config pool:

a. View the machine config pool:

```
$ oc describe machineconfigpool <name>
```

For example:

```
$ oc describe machineconfigpool worker
```

Example output

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: 2019-02-08T14:52:39Z
  generation: 1
  labels:
    custom-kubelet: set-max-pods 1
```

1   If a label has been added it appears under **labels**.

b. If the label is not present, add a key/value pair:

```
$ oc label machineconfigpool worker custom-kubelet=set-max-pods
```

Procedure

1. View the available machine configuration objects that you can select:

```
$ oc get machineconfig
```

By default, the two kubelet-related configs are **01-master-kubelet** and **01-worker-kubelet**.

2. Check the current value for the maximum pods per node:

```
$ oc describe node <node_name>
```
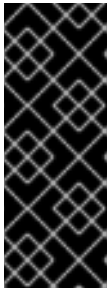
For example:

```
$ oc describe node ci-ln-5grqprb-f76d1-ncnqq-worker-a-mdv94
```

Look for **value: pods: <value>** in the **Allocatable** stanza:

Example output

```
Allocatable:
 attachable-volumes-aws-ebs:  25
 cpu:                3500m
 hugepages-1Gi:          0
 hugepages-2Mi:          0
 memory:             15341844Ki
 pods:              250
```

3. Set the maximum pods per node on the worker nodes by creating a custom resource file that contains the kubelet configuration:

> **IMPORTANT**
>
> Kubelet configurations that target a specific machine config pool also affect any dependent pools. For example, creating a kubelet configuration for the pool containing worker nodes will also apply to any subset pools, including the pool containing infrastructure nodes. To avoid this, you must create a new machine config pool with a selection expression that only includes worker nodes, and have your kubelet configuration target this new pool.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods  1
  kubeletConfig:
    maxPods: 500  2
```

1 Enter the label from the machine config pool.

2 Add the kubelet configuration. In this example, use **maxPods** to set the maximum pods per node.

> **NOTE**
>
> The rate at which the kubelet talks to the API server depends on queries per second (QPS) and burst values. The default values, **50** for **kubeAPIQPS** and **100** for **kubeAPIBurst**, are sufficient if there are limited pods running on each node. It is recommended to update the kubelet QPS and burst rates if there are enough CPU and memory resources on the node.
>
> ```
> apiVersion: machineconfiguration.openshift.io/v1
> kind: KubeletConfig
> metadata:
>   name: set-max-pods
> spec:
>   machineConfigPoolSelector:
>     matchLabels:
>       custom-kubelet: set-max-pods
>   kubeletConfig:
>     maxPods: <pod_count>
>     kubeAPIBurst: <burst_rate>
>     kubeAPIQPS: <QPS>
> ```

   a. Update the machine config pool for workers with the label:

   ```
   $ oc label machineconfigpool worker custom-kubelet=set-max-pods
   ```

b. Create the **KubeletConfig** object:

```
$ oc create -f change-maxPods-cr.yaml
```

c. Verify that the **KubeletConfig** object is created:

```
$ oc get kubeletconfig
```

Example output

```
NAME            AGE
set-max-pods    15m
```

Depending on the number of worker nodes in the cluster, wait for the worker nodes to be rebooted one by one. For a cluster with 3 worker nodes, this could take about 10 to 15 minutes.

4. Verify that the changes are applied to the node:

a. Check on a worker node that the **maxPods** value changed:

```
$ oc describe node <node_name>
```

b. Locate the **Allocatable** stanza:

```
...
Allocatable:
  attachable-volumes-gce-pd:  127
  cpu:                3500m
  ephemeral-storage:      123201474766
  hugepages-1Gi:          0
  hugepages-2Mi:          0
  memory:             14225400Ki
  pods:               500 ❶
...
```

❶ In this example, the **pods** parameter should report the value you set in the **KubeletConfig** object.

5. Verify the change in the **KubeletConfig** object:

```
$ oc get kubeletconfigs set-max-pods -o yaml
```

This should show a status of **True** and **type:Success**, as shown in the following example:

```
spec:
  kubeletConfig:
    maxPods: 500
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods
status:
  conditions:
```

```
- lastTransitionTime: "2021-06-30T17:04:07Z"
  message: Success
  status: "True"
  type: Success
```

4.2. CREATING A CONTAINERRUNTIMECONFIG CR TO EDIT CRI-O PARAMETERS

You can change some of the settings associated with the OpenShift Container Platform CRI-O runtime for the nodes associated with a specific machine config pool (MCP). Using a **ContainerRuntimeConfig** custom resource (CR), you set the configuration values and add a label to match the MCP. The MCO then rebuilds the **crio.conf** and **storage.conf** configuration files on the associated nodes with the updated values.

> **NOTE**
>
> To revert the changes implemented by using a **ContainerRuntimeConfig** CR, you must delete the CR. Removing the label from the machine config pool does not revert the changes.

You can modify the following settings by using a **ContainerRuntimeConfig** CR:

- **PIDs limit**: Setting the PIDs limit in the **ContainerRuntimeConfig** is expected to be deprecated. If PIDs limits are required, it is recommended to use the **podPidsLimit** field in the **KubeletConfig** CR instead. The default value of the **podPidsLimit** field is **4096**.

  > **NOTE**
  >
  > The CRI-O flag is applied on the cgroup of the container, while the Kubelet flag is set on the cgroup of the pod. Please adjust the PIDs limit accordingly.

- **Log level**: The **logLevel** parameter sets the CRI-O **log_level** parameter, which is the level of verbosity for log messages. The default is **info** (**log_level = info**). Other options include **fatal**, **panic**, **error**, **warn**, **debug**, and **trace**.

- **Overlay size**: The **overlaySize** parameter sets the CRI-O Overlay storage driver **size** parameter, which is the maximum size of a container image.

- **Maximum log size**: Setting the maximum log size in the **ContainerRuntimeConfig** is expected to be deprecated. If a maximum log size is required, it is recommended to use the **containerLogMaxSize** field in the **KubeletConfig** CR instead.

- **Container runtime**: The **defaultRuntime** parameter sets the container runtime to either **runc** or **crun**. The default is **runc**.

You should have one **ContainerRuntimeConfig** CR for each machine config pool with all the config changes you want for that pool. If you are applying the same content to all the pools, you only need one **ContainerRuntimeConfig** CR for all the pools.

You should edit an existing **ContainerRuntimeConfig** CR to modify existing settings or add new settings instead of creating a new CR for each change. It is recommended to create a new **ContainerRuntimeConfig** CR only to modify a different machine config pool, or for changes that are intended to be temporary so that you can revert the changes.

You can create multiple **ContainerRuntimeConfig** CRs, as needed, with a limit of 10 per cluster. For the first **ContainerRuntimeConfig** CR, the MCO creates a machine config appended with **containerruntime**. With each subsequent CR, the controller creates a new **containerruntime** machine config with a numeric suffix. For example, if you have a **containerruntime** machine config with a **-2** suffix, the next **containerruntime** machine config is appended with **-3**.

If you want to delete the machine configs, you should delete them in reverse order to avoid exceeding the limit. For example, you should delete the **containerruntime-3** machine config before deleting the **containerruntime-2** machine config.

> **NOTE**
>
> If you have a machine config with a **containerruntime-9** suffix, and you create another **ContainerRuntimeConfig** CR, a new machine config is not created, even if there are fewer than 10 **containerruntime** machine configs.

Example showing multiple ContainerRuntimeConfig CRs

```
$ oc get ctrcfg
```

Example output

```
NAME        AGE
ctr-overlay  15m
ctr-level    5m45s
```

Example showing multiple containerruntime machine configs

```
$ oc get mc | grep container
```

Example output

```
...
01-master-container-runtime                    b5c5119de007945b6fe6fb215db3b8e2ceb12511  3.2.0
57m
...
01-worker-container-runtime                    b5c5119de007945b6fe6fb215db3b8e2ceb12511  3.2.0
57m
...
99-worker-generated-containerruntime           b5c5119de007945b6fe6fb215db3b8e2ceb12511
3.2.0          26m
99-worker-generated-containerruntime-1          b5c5119de007945b6fe6fb215db3b8e2ceb12511
3.2.0          17m
99-worker-generated-containerruntime-2          b5c5119de007945b6fe6fb215db3b8e2ceb12511
3.2.0          7m26s
...
```

The following example sets the **log_level** field to **debug** and sets the overlay size to 8 GB:

Example ContainerRuntimeConfig CR

```
apiVersion: machineconfiguration.openshift.io/v1
```

```
kind: ContainerRuntimeConfig
metadata:
 name: overlay-size
spec:
 machineConfigPoolSelector:
  matchLabels:
    pools.operator.machineconfiguration.openshift.io/worker: "  1
 containerRuntimeConfig:
  logLevel: debug  2
  overlaySize: 8G  3
  defaultRuntime: "crun"  4
```

1 Specifies the machine config pool label. For a container runtime config, the role must match the name of the associated machine config pool.

2 Optional: Specifies the level of verbosity for log messages.

3 Optional: Specifies the maximum size of a container image.

4 Optional: Specifies the container runtime to deploy to new containers. The default value is **runc**.

Procedure

To change CRI-O settings using the **ContainerRuntimeConfig** CR:

1. Create a YAML file for the **ContainerRuntimeConfig** CR:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: ContainerRuntimeConfig
   metadata:
    name: overlay-size
   spec:
    machineConfigPoolSelector:
     matchLabels:
       pools.operator.machineconfiguration.openshift.io/worker: "  1
    containerRuntimeConfig:  2
     logLevel: debug
     overlaySize: 8G
   ```

   1 Specify a label for the machine config pool that you want you want to modify.

   2 Set the parameters as needed.

2. Create the **ContainerRuntimeConfig** CR:

   ```
   $ oc create -f <file_name>.yaml
   ```

3. Verify that the CR is created:

   ```
   $ oc get ContainerRuntimeConfig
   ```

   Example output

```
NAME         AGE
overlay-size   3m19s
```

4. Check that a new **containerruntime** machine config is created:

```
$ oc get machineconfigs | grep containerrun
```

Example output

```
99-worker-generated-containerruntime   2c9371fbb673b97a6fe8b1c52691999ed3a1bfc2
3.2.0  31s
```

5. Monitor the machine config pool until all are shown as ready:

```
$ oc get mcp worker
```

Example output

```
NAME    CONFIG           UPDATED  UPDATING  DEGRADED  MACHINECOUNT
READYMACHINECOUNT  UPDATEDMACHINECOUNT  DEGRADEDMACHINECOUNT
AGE
worker   rendered-worker-169  False    True     False    3          1            1                0
9h
```

6. Verify that the settings were applied in CRI-O:

    a. Open an **oc debug** session to a node in the machine config pool and run **chroot /host**.

    ```
    $ oc debug node/<node_name>
    ```

    ```
    sh-4.4# chroot /host
    ```

    b. Verify the changes in the **crio.conf** file:

    ```
    sh-4.4# crio config | grep 'log_level'
    ```

    Example output

    ```
    log_level = "debug"
    ```

    c. Verify the changes in the `storage.conf` file:

    ```
    sh-4.4# head -n 7 /etc/containers/storage.conf
    ```

    Example output

    ```
    [storage]
      driver = "overlay"
      runroot = "/var/run/containers/storage"
      graphroot = "/var/lib/containers/storage"
    ```

```
[storage.options]
  additionalimagestores = []
  size = "8G"
```

4.3. SETTING THE DEFAULT MAXIMUM CONTAINER ROOT PARTITION SIZE FOR OVERLAY WITH CRI-O

The root partition of each container shows all of the available disk space of the underlying host. Follow this guidance to set a maximum partition size for the root disk of all containers.

To configure the maximum Overlay size, as well as other CRI-O options like the log level, you can create the following **ContainerRuntimeConfig** custom resource definition (CRD):

```
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
 name: overlay-size
spec:
 machineConfigPoolSelector:
   matchLabels:
     custom-crio: overlay-size
 containerRuntimeConfig:
   logLevel: debug
   overlaySize: 8G
```

Procedure

1. Create the configuration object:

   ```
   $ oc apply -f overlaysize.yml
   ```

2. To apply the new CRI-O configuration to your worker nodes, edit the worker machine config pool:

   ```
   $ oc edit machineconfigpool worker
   ```

3. Add the **custom-crio** label based on the **matchLabels** name you set in the **ContainerRuntimeConfig** CRD:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfigPool
   metadata:
     creationTimestamp: "2020-07-09T15:46:34Z"
     generation: 3
     labels:
       custom-crio: overlay-size
       machineconfiguration.openshift.io/mco-built-in: ""
   ```

4. Save the changes, then view the machine configs:

   ```
   $ oc get machineconfigs
   ```

   New **99-worker-generated-containerruntime** and **rendered-worker-xyz** objects are created:

Example output

```
99-worker-generated-containerruntime  4173030d89fbf4a7a0976d1665491a4d9a6e54f1
3.2.0          7m42s
rendered-worker-xyz               4173030d89fbf4a7a0976d1665491a4d9a6e54f1  3.2.0
7m36s
```

5. After those objects are created, monitor the machine config pool for the changes to be applied:

```
$ oc get mcp worker
```

The worker nodes show **UPDATING** as **True**, as well as the number of machines, the number updated, and other details:

Example output

```
NAME   CONFIG          UPDATED  UPDATING  DEGRADED  MACHINECOUNT
READYMACHINECOUNT  UPDATEDMACHINECOUNT   DEGRADEDMACHINECOUNT
AGE
worker rendered-worker-xyz False True False    3         2          2          0
20h
```

When complete, the worker nodes transition back to **UPDATING** as **False**, and the **UPDATEDMACHINECOUNT** number matches the **MACHINECOUNT**:

Example output

```
NAME   CONFIG          UPDATED  UPDATING  DEGRADED  MACHINECOUNT
READYMACHINECOUNT  UPDATEDMACHINECOUNT   DEGRADEDMACHINECOUNT
AGE
worker  rendered-worker-xyz  True    False    False    3    3      3      0
20h
```

Looking at a worker machine, you see that the new 8 GB max size configuration is applied to all of the workers:

Example output

```
head -n 7 /etc/containers/storage.conf
[storage]
  driver = "overlay"
  runroot = "/var/run/containers/storage"
  graphroot = "/var/lib/containers/storage"
  [storage.options]
    additionalimagestores = []
    size = "8G"
```

Looking inside a container, you see that the root partition is now 8 GB:

Example output

```
~ $ df -h
Filesystem              Size    Used Available Use% Mounted on
overlay                 8.0G    8.0K    8.0G   0% /
```

CHAPTER 5. UPDATED BOOT IMAGES

The Machine Config Operator (MCO) uses a boot image to start a Red Hat Enterprise Linux CoreOS (RHCOS) node. By default, OpenShift Container Platform does not manage the boot image.

This means that the boot image in your cluster is not updated along with your cluster. For example, if your cluster was originally created with OpenShift Container Platform 4.12, the boot image that the cluster uses to create nodes is the same 4.12 version, even if your cluster is at a later version. If the cluster is later upgraded to 4.13 or later, new nodes continue to scale with the same 4.12 image.

This process could cause the following issues:

- Extra time to start nodes

- Certificate expiration issues

- Version skew issues

To avoid these issues, you can configure your cluster to update the boot image whenever you update your cluster. By modifying the **MachineConfiguration** object, you can enable this feature. Currently, the ability to update the boot image is available for only Google Cloud Platform (GCP) clusters and is not supported for clusters managed by the Cluster API.

> **IMPORTANT**
>
> The updating boot image feature is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.
>
> For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

To view the current boot image used in your cluster, examine a machine set:

Example machine set with the boot image reference

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: ci-ln-hmy310k-72292-5f87z-worker-a
  namespace: openshift-machine-api
spec:
# ...
  template:
# ...
    spec:
# ...
      providerSpec:
# ...
        value:
          disks:
          - autoDelete: true
```

```
boot: true
image: projects/rhcos-cloud/global/images/rhcos-412-85-202203181601-0-gcp-x86-64 ❶
# ...
```

❶ This boot image is the same as the originally-installed OpenShift Container Platform version, in this example OpenShift Container Platform 4.12, regardless of the current version of the cluster. The way that the boot image is represented in the machine set depends on the platform, as the structure of the **providerSpec** field differs from platform to platform.

If you configure your cluster to update your boot images, the boot image referenced in your machine sets matches the current version of the cluster.

5.1. CONFIGURING UPDATED BOOT IMAGES

By default, OpenShift Container Platform does not manage the boot image. You can configure your cluster to update the boot image whenever you update your cluster by modifying the **MachineConfiguration** object.

Prerequisites

- You have enabled the **TechPreviewNoUpgrade** feature set by using the feature gates. For more information, see "Enabling features using feature gates" in the *Additional resources* section.

Procedure

1. Edit the **MachineConfiguration** object, named **cluster**, to enable the updating of boot images by running the following command:

   ```
   $ oc edit MachineConfiguration cluster
   ```

   a. Optional: Configure the boot image update feature for all the machine sets:

   ```
   apiVersion: operator.openshift.io/v1
   kind: MachineConfiguration
   metadata:
     name: cluster
     namespace: openshift-machine-config-operator
   spec:
   # ...
     managedBootImages: ❶
       machineManagers:
       - resource: machinesets
         apiGroup: machine.openshift.io
         selection:
           mode: All ❷
   ```

   ❶ Activates the boot image update feature.

   ❷ Specifies that all the machine sets in the cluster are to be updated.

   b. Optional: Configure the boot image update feature for specific machine sets:
```

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  name: cluster
  namespace: openshift-machine-config-operator
spec:
# ...
  managedBootImages: ❶
    machineManagers:
    - resource: machinesets
      apiGroup: machine.openshift.io
      selection:
        mode: Partial
          partial:
            machineResourceSelector:
              matchLabels:
                update-boot-image: "true" ❷
```

❶ Activates the boot image update feature.

❷ Specifies that any machine set with this label is to be updated.

TIP

If an appropriate label is not present on the machine set, add a key/value pair by running a command similar to following:

```
$ oc label machineset.machine ci-ln-hmy310k-72292-5f87z-worker-a update-boot-image=true -n openshift-machine-api
```

Verification

1. Get the boot image version by running the following command:

   ```
   $ oc get machinesets <machineset_name> -n openshift-machine-api -o yaml
   ```

   Example machine set with the boot image reference

   ```
   apiVersion: machine.openshift.io/v1beta1
   kind: MachineSet
   metadata:
     labels:
       machine.openshift.io/cluster-api-cluster: ci-ln-77hmkpt-72292-d4pxp
       update-boot-image: "true"
     name: ci-ln-77hmkpt-72292-d4pxp-worker-a
     namespace: openshift-machine-api
   spec:
   # ...
     template:
   # ...
       spec:
   # ...
         providerSpec:
   ```

```
# ...
    value:
     disks:
     - autoDelete: true
       boot: true
       image: projects/rhcos-cloud/global/images/rhcos-416-92-202402201450-0-gcp-x86-
64 1
# ...
```

1　This boot image is the same as the current OpenShift Container Platform version.

Additional resources

- [Enabling features using feature gates](#)

5.2. DISABLING UPDATED BOOT IMAGES

To disable the updated boot image feature, edit the **MachineConfiguration** object to remove the **managedBootImages** stanza.

If you disable this feature after some nodes have been created with the new boot image version, any existing nodes retain their current boot image. Turning off this feature does not rollback the nodes or machine sets to the originally–installed boot image. The machine sets retain the boot image version that was present when the feature was enabled and is not updated again when the cluster is upgraded to a new OpenShift Container Platform version in the future.

Procedure

1. Disable updated boot images by editing the **MachineConfiguration** object:

   ```
   $ oc edit MachineConfiguration cluster
   ```

2. Remove the **managedBootImages** stanza:

   ```
   apiVersion: operator.openshift.io/v1
   kind: MachineConfiguration
   metadata:
     name: cluster
     namespace: openshift-machine-config-operator
   spec:
   # ...
     managedBootImages: 1
       machineManagers:
       - resource: machinesets
         apiGroup: machine.openshift.io
         selection:
           mode: All
   ```

1　Remove the entire stanza to disable updated boot images.

CHAPTER 6. MANAGING UNUSED RENDERED MACHINE CONFIGS

The Machine Config Operator (MCO) does not perform any garbage collection activities. This means that all rendered machine configs remain in the cluster. Each time a user or controller applies a new machine config, the MCO creates new rendered configs for each affected machine config pool. Over time, this can lead to a large number of rendered machine configs, which can make working with machine configs confusing. Having too many rendered machine configs can also contribute to disk space issues and performance issues with etcd.

You can remove old, unused rendered machine configs by using the **oc adm prune renderedmachineconfigs** command with the **--confirm** flag. With this command, you can remove all unused rendered machine configs or only those in a specific machine config pool. You can also remove a specified number of unused rendered machine configs in order to keep some older machine configs, in case you want to check older configurations.

You can use the **oc adm prune renderedmachineconfigs** command without the **--confirm** flag to see which rendered machine configs would be removed.

Use the **list** subcommand to display all the rendered machine configs in the cluster or a specific machine config pool.

> **NOTE**
>
> The **oc adm prune renderedmachineconfigs** command deletes only rendered machine configs that are not in use. If a rendered machine configs are in use by a machine config pool, the rendered machine config is not deleted. In this case, the command output specifies the reason that the rendered machine config was not deleted.

6.1. VIEWING RENDERED MACHINE CONFIGS

You can view a list of rendered machine configs by using the **oc adm prune renderedmachineconfigs** command with the **list** subcommand.

For example, the command in the following procedure would list all rendered machine configs for the **worker** machine config pool.

Procedure

- Optional: List the rendered machine configs by using the following command:

  ```
  $ oc adm prune renderedmachineconfigs list --in-use=false --pool-name=worker
  ```

  where:

  list

  Displays a list of rendered machine configs in your cluster.

  --in-use

  Optional: Specifies whether to display only the used machine configs or all machine configs from the specified pool. If **true**, the output lists the rendered machine configs that are being used by a machine config pool. If **false**, the output lists all rendered machine configs in the cluster. The default value is **false**.

  --pool-name

Optional: Specifies the machine config pool from which to display the machine configs.

Example output

```
worker
status: rendered-worker-ae115e2b5e6ae05e0e6e5d62c7d0dd81
spec: rendered-worker-ae115e2b5e6ae05e0e6e5d62c7d0dd81
```

- List the rendered machine configs that you can remove automatically by running the following command. Any rendered machine config marked with **as it's currently in use** in the command output is not removed.

```
$ oc adm prune renderedmachineconfigs --pool-name=worker
```

The command runs in dry-run mode, and no machine configs are removed.

where:

--pool-name

Optional: Displays the machine configs in the specified machine config pool.

Example output

```
Dry run enabled - no modifications will be made. Add --confirm to remove rendered machine
configs.
DRY RUN: Deleted rendered MachineConfig rendered-worker-
23d7322831a57f02998e7e1600a0865f
DRY RUN: Deleted rendered MachineConfig rendered-worker-
fc94397dc7c43808c7014683c208956e
DRY RUN: Skipping deletion of rendered MachineConfig rendered-worker-
ad5a3cad36303c363cf458ab0524e7c0 as it's currently in use
```

6.2. REMOVING UNUSED RENDERED MACHINE CONFIGS

You can remove unused rendered machine configs by using the **oc adm prune renderedmachineconfigs** command with the **--confirm** command. If any rendered machine config is not deleted, the command output indicates which was not deleted and lists the reason for skipping the deletion.

Procedure

1. Optional: List the rendered machine configs that you can remove automatically by running the following command. Any rendered machine config marked with **as it's currently in use** in the command output is not removed.

```
$ oc adm prune renderedmachineconfigs --pool-name=worker
```

Example output

```
Dry run enabled - no modifications will be made. Add --confirm to remove rendered machine
configs.
DRY RUN: Deleted rendered MachineConfig rendered-worker-
23d7322831a57f02998e7e1600a0865f
```

> DRY RUN: Deleted rendered MachineConfig rendered-worker-
> fc94397dc7c43808c7014683c208956e
> DRY RUN: Skipping deletion of rendered MachineConfig rendered-worker-
> ad5a3cad36303c363cf458ab0524e7c0 as it's currently in use

where:

pool-name

Optional: Specifies the machine config pool where you want to delete the machine configs from.

2. Remove the unused rendered machine configs by running the following command. The command in the following procedure would delete the two oldest unused rendered machine configs in the **worker** machine config pool.

> $ oc adm prune renderedmachineconfigs --pool-name=worker --count=2 --confirm

where:

--count

Optional: Specifies the maximum number of unused rendered machine configs you want to delete, starting with the oldest.

--confirm

Optional: Specifies the maximum number of unused rendered machine configs you want to delete, starting with the oldest.

--pool-name

Optional: Specifies the machine config pool from which you want to delete the machine. If not specified, all the pools are evaluated.

CHAPTER 7. RHCOS IMAGE LAYERING

Red Hat Enterprise Linux CoreOS (RHCOS) image layering allows you to easily extend the functionality of your base RHCOS image by *layering* additional images onto the base image. This layering does not modify the base RHCOS image. Instead, it creates a *custom layered image* that includes all RHCOS functionality and adds additional functionality to specific nodes in the cluster.

7.1. ABOUT RHCOS IMAGE LAYERING

Image layering allows you to customize the underlying node operating system on any of your cluster worker nodes. This helps keep everything up-to-date, including the node operating system and any added customizations such as specialized software.

You create a custom layered image by using a Containerfile and applying it to nodes by using a custom object. At any time, you can remove the custom layered image by deleting that custom object.

With RHCOS image layering, you can install RPMs into your base image, and your custom content will be booted alongside RHCOS. The Machine Config Operator (MCO) can roll out these custom layered images and monitor these custom containers in the same way it does for the default RHCOS image. RHCOS image layering gives you greater flexibility in how you manage your RHCOS nodes.

> **IMPORTANT**
>
> Installing realtime kernel and extensions RPMs as custom layered content is not recommended. This is because these RPMs can conflict with RPMs installed by using a machine config. If there is a conflict, the MCO enters a **degraded** state when it tries to install the machine config RPM. You need to remove the conflicting extension from your machine config before proceeding.

As soon as you apply the custom layered image to your cluster, you effectively *take ownership* of your custom layered images and those nodes. While Red Hat remains responsible for maintaining and updating the base RHCOS image on standard nodes, you are responsible for maintaining and updating images on nodes that use a custom layered image. You assume the responsibility for the package you applied with the custom layered image and any issues that might arise with the package.

There are two methods for deploying a custom layered image onto your nodes:

On-cluster layering

With on-cluster layering, you create a **MachineOSConfig** object where you include the Containerfile and other parameters. The build is performed on your cluster and the resulting custom layered image is automatically pushed to your repository and applied to the machine config pool that you specified in the **MachineOSConfig** object. The entire process is performed completely within your cluster.

> **IMPORTANT**
>
> On-cluster image layering is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.
>
> For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Out-of-cluster layering

With [out-of-cluster layering](#), you create a Containerfile that references an OpenShift Container Platform image and the RPM that you want to apply, build the layered image in your own environment, and push the image to your repository. Then, in your cluster, create a **MachineConfig** object for the targeted node pool that points to the new image. The Machine Config Operator overrides the base RHCOS image, as specified by the **osImageURL** value in the associated machine config, and boots the new image.

> **IMPORTANT**
>
> For both methods, use the same base RHCOS image installed on the rest of your cluster. Use the **oc adm release info --image-for rhel-coreos** command to obtain the base image used in your cluster.

7.2. EXAMPLE CONTAINERFILES

RHCOS image layering allows you to use the following types of images to create custom layered images:

- **OpenShift Container Platform Hotfixes**. You can work with Customer Experience and Engagement (CEE) to obtain and apply [Hotfix packages](#) on top of your RHCOS image. In some instances, you might want a bug fix or enhancement before it is included in an official OpenShift Container Platform release. RHCOS image layering allows you to easily add the Hotfix before it is officially released and remove the Hotfix when the underlying RHCOS image incorporates the fix.

  > **IMPORTANT**
  >
  > Some Hotfixes require a Red Hat Support Exception and are outside of the normal scope of OpenShift Container Platform support coverage or life cycle policies.

  Hotfixes are provided to you based on [Red Hat Hotfix policy](#). Apply it on top of the base image and test that new custom layered image in a non-production environment. When you are satisfied that the custom layered image is safe to use in production, you can roll it out on your own schedule to specific node pools. For any reason, you can easily roll back the custom layered image and return to using the default RHCOS.

  Example on-cluster Containerfile to apply a Hotfix

  ```
  # Using a 4.12.0 image
  containerfileArch: noarch
  content: |-
    FROM configs AS final
    #Install hotfix rpm
    RUN rpm-ostree override replace https://example.com/myrepo/haproxy-1.0.16-5.el8.src.rpm && \
        rpm-ostree cleanup -m && \
        ostree container commit
  ```

  Example out-of-cluster Containerfile to apply a Hotfix

  ```
  # Using a 4.12.0 image
  ```

```
FROM quay.io/openshift-release-dev/ocp-release@sha256...
#Install hotfix rpm
RUN rpm-ostree override replace https://example.com/myrepo/haproxy-1.0.16-5.el8.src.rpm
&& \
    rpm-ostree cleanup -m && \
    ostree container commit
```

- **RHEL packages**. You can download Red Hat Enterprise Linux (RHEL) packages from the Red Hat Customer Portal, such as chrony, firewalld, and iputils.

Example out-of-cluster Containerfile to apply the libreswan utility

```
# Get RHCOS base image of target cluster `oc adm release info --image-for rhel-coreos`
# hadolint ignore=DL3006
FROM quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256...

# Install our config file
COPY my-host-to-host.conf /etc/ipsec.d/

# RHEL entitled host is needed here to access RHEL packages
# Install libreswan as extra RHEL package
RUN rpm-ostree install libreswan && \
    systemctl enable ipsec && \
    ostree container commit
```

Because libreswan requires additional RHEL packages, the image must be built on an entitled RHEL host. For RHEL entitlements to work, you must copy the **etc-pki-entitlement** secret into the **openshift-machine-api** namespace.

- **Third-party packages**. You can download and install RPMs from third-party organizations, such as the following types of packages:

  - Bleeding edge drivers and kernel enhancements to improve performance or add capabilities.

  - Forensic client tools to investigate possible and actual break-ins.

  - Security agents.

  - Inventory agents that provide a coherent view of the entire cluster.

  - SSH Key management packages.

Example on-cluster Containerfile to apply a third-party package from EPEL

```
FROM configs AS final

#Enable EPEL (more info at https://docs.fedoraproject.org/en-US/epel/ ) and install htop
RUN rpm-ostree install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
&& \
    rpm-ostree install htop && \
    ostree container commit
```

Example out-of-cluster Containerfile to apply a third-party package from EPEL

```
# Get RHCOS base image of target cluster `oc adm release info --image-for rhel-coreos`
# hadolint ignore=DL3006
FROM quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256...

# Install our config file
COPY my-host-to-host.conf /etc/ipsec.d/

# RHEL entitled host is needed here to access RHEL packages
# Install libreswan as extra RHEL package
RUN rpm-ostree install libreswan && \
    systemctl enable ipsec && \
    ostree container commit
```

This Containerfile installs the RHEL fish program. Because fish requires additional RHEL packages, the image must be built on an entitled RHEL host. For RHEL entitlements to work, you must copy the **etc-pki-entitlement** secret into the **openshift-machine-api** namespace.

Example on-cluster Containerfile to apply a third-party package that has RHEL dependencies

```
FROM configs AS final

# RHEL entitled host is needed here to access RHEL packages
# Install fish as third party package from EPEL
RUN rpm-ostree install
https://dl.fedoraproject.org/pub/epel/9/Everything/x86_64/Packages/f/fish-3.3.1-
3.el9.x86_64.rpm && \
    ostree container commit
```

Example out-of-cluster Containerfile to apply a third-party package that has RHEL dependencies

```
# Get RHCOS base image of target cluster `oc adm release info --image-for rhel-coreos`
# hadolint ignore=DL3006
FROM quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256...

# Install our config file
COPY my-host-to-host.conf /etc/ipsec.d/

# RHEL entitled host is needed here to access RHEL packages
# Install libreswan as extra RHEL package
RUN rpm-ostree install libreswan && \
    systemctl enable ipsec && \
    ostree container commit
```

After you create the machine config, the Machine Config Operator (MCO) performs the following steps:

1. Renders a new machine config for the specified pool or pools.

2. Performs cordon and drain operations on the nodes in the pool or pools.

3. Writes the rest of the machine config parameters onto the nodes.

4. Applies the custom layered image to the node.

5. Reboots the node using the new image.

> **IMPORTANT**
>
> It is strongly recommended that you test your images outside of your production environment before rolling out to your cluster.

7.3. USING ON-CLUSTER LAYERING TO APPLY A CUSTOM LAYERED IMAGE

To apply a custom layered image to your cluster by using the on-cluster build process, make a **MachineOSConfig** custom resource that includes a Containerfile, a machine config pool reference, repository push and pull secrets, and other parameters as described in the prerequisites.

When you create the object, the Machine Config Operator (MCO) creates a **MachineOSBuild** object and a **machine-os-builder** pod. The build process also creates transient objects, such as config maps, which are cleaned up after the build is complete.

When the build is complete, the MCO pushes the new custom layered image to your repository for use when deploying new nodes. You can see the digested image pull spec for the new custom layered image in the **MachineOSBuild** object and **machine-os-builder** pod.

You should not need to interact with these new objects or the **machine-os-builder** pod. However, you can use all of these resources for troubleshooting, if necessary.

You need a separate **MachineOSConfig** CR for each machine config pool where you want to use a custom layered image.

> **IMPORTANT**
>
> On-cluster image layering is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.
>
> For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Prerequisites

- You have enabled the **TechPreviewNoUpgrade** feature set by using the feature gates. For more information, see "Enabling features using feature gates".

- You have the pull secret in the **openshift-machine-config-operator** namespace that the MCO needs to pull the base operating system image.

- You have the push secret that the MCO needs to push the new custom layered image to your registry.

- You have a pull secret that your nodes need to pull the new custom layered image from your registry. This should be a different secret than the one used to push the image to the repository.

- You are familiar with how to configure a Containerfile. Instructions on how to create a Containerfile are beyond the scope of this documentation.

- Optional: You have a separate machine config pool for the nodes where you want to apply the custom layered image.

Procedure

1. Create a **machineOSconfig** object:

   a. Create a YAML file similar to the following:

   ```
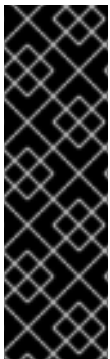   apiVersion: machineconfiguration.openshift.io/v1alpha1
   kind: MachineOSConfig
   metadata:
     name: layered
   spec:
     machineConfigPool:
       name: <mcp_name> 1
     buildInputs:
       containerFile: 2
       - containerfileArch: noarch
         content: |-
           FROM configs AS final
           RUN rpm-ostree install cowsay && \
             ostree container commit
       imageBuilder: 3
         imageBuilderType: PodImageBuilder
       baseImagePullSecret: 4
         name: global-pull-secret-copy
       renderedImagePushspec: image-registry.openshift-image-
   registry.svc:5000/openshift/os-image:latest 5
       renderedImagePushSecret: 6
         name: builder-dockercfg-7lzwl
     buildOutputs: 7
       currentImagePullSecret:
         name: builder-dockercfg-7lzwl
   ```

   [1] Specifies the name of the machine config pool associated with the nodes where you want to deploy the custom layered image.

   [2] Specifies the Containerfile to configure the custom layered image.

   [3] Specifies the name of the image builder to use. This must be **PodImageBuilder**.

   [4] Specifies the name of the pull secret that the MCO needs to pull the base operating system image from the registry.

   [5] Specifies the image registry to push the newly-built custom layered image to. This can be any registry that your cluster has access to. This example uses the internal OpenShift Container Platform registry.

   [6] Specifies the name of the push secret that the MCO needs to push the newly-built custom layered image to that registry.

   [7] Specifies the secret required by the image registry that the nodes need to pull the newly-built custom layered image. This should be a different secret than the one used to push the image to your repository.

b. Create the **MachineOSConfig** object:

```
$ oc create -f <file_name>.yaml
```

2. If necessary, when the **MachineOSBuild** object has been created and is in the **READY** state, modify the node spec for the nodes where you want to use the new custom layered image:

a. Check that the **MachineOSBuild** object is **READY**. When the **SUCCEEDED** value is **True**, the build is complete.

```
$ oc get machineosbuild
```

Example output showing that the MachineOSBuild object is ready

```
NAME                                                  PREPARED  BUILDING  SUCCEEDED
INTERRUPTED  FAILED
layered-rendered-layered-ad5a3cad36303c363cf458ab0524e7c0-builder  False
False    True     False     False
```

b. Edit the nodes where you want to deploy the custom layered image by adding a label for the machine config pool you specified in the **MachineOSConfig** object:

```
$ oc label node <node_name> 'node-role.kubernetes.io/<mcp_name>='
```

where:

node-role.kubernetes.io/<mcp_name>=

Specifies a node selector that identifies the nodes to deploy the custom layered image.

When you save the changes, the MCO drains, cordons, and reboots the nodes. After the reboot, the node will be using the new custom layered image.

Verification

1. Verify that the new pods are running by using the following command:

```
$ oc get pods -n <machineosbuilds_namespace>
```

```
NAME                                             READY  STATUS    RESTARTS  AGE
build-rendered-layered-ad5a3cad36303c363cf458ab0524e7c0        2/2    Running   0
2m40s  ❶
# ...
machine-os-builder-6fb66cfb99-zcpvq                        1/1    Running   0     2m42s  ❷
```

❶ This is the build pod where the custom layered image is building.

❷ This pod can be used for troubleshooting.

2. Verify that the **MachineOSConfig** object contains a reference to the new custom layered image:

```
$ oc describe MachineOSConfig <object_name>
```

```
apiVersion: machineconfiguration.openshift.io/v1alpha1
kind: MachineOSConfig
metadata:
  name: layered
spec:
  buildInputs:
    baseImagePullSecret:
      name: global-pull-secret-copy
    containerFile:
    - containerfileArch: noarch
      content: ""
    imageBuilder:
      imageBuilderType: PodImageBuilder
    renderedImagePushSecret:
      name: builder-dockercfg-ng82t-canonical
    renderedImagePushspec: image-registry.openshift-image-registry.svc:5000/openshift-
machine-config-operator/os-image:latest
  buildOutputs:
    currentImagePullSecret:
      name: global-pull-secret-copy
  machineConfigPool:
    name: layered
status:
  currentImagePullspec: image-registry.openshift-image-registry.svc:5000/openshift-machine-
config-operator/os-
image@sha256:f636fa5b504e92e6faa22ecd71a60b089dab72200f3d130c68dfec07148d11cd
  ❶
```

❶ Digested image pull spec for the new custom layered image.

3. Verify that the **MachineOSBuild** object contains a reference to the new custom layered image.

```
$ oc describe machineosbuild <object_name>
```

```
apiVersion: machineconfiguration.openshift.io/v1alpha1
kind: MachineOSBuild
metadata:
  name: layered-rendered-layered-ad5a3cad36303c363cf458ab0524e7c0-builder
spec:
  desiredConfig:
    name: rendered-layered-ad5a3cad36303c363cf458ab0524e7c0
  machineOSConfig:
    name: layered
  renderedImagePushspec: image-registry.openshift-image-registry.svc:5000/openshift-
machine-config-operator/os-image:latest
# ...
status:
  conditions:
    - lastTransitionTime: "2024-05-21T20:25:06Z"
      message: Build Ready
      reason: Ready
      status: "True"
      type: Succeeded
  finalImagePullspec: image-registry.openshift-image-registry.svc:5000/openshift-machine-
```

```
config-operator/os-
image@sha256:f636fa5b504e92e6faa22ecd71a60b089dab72200f3d130c68dfec07148d11cd
```
1

1 Digested image pull spec for the new custom layered image.

4. Verify that the appropriate nodes are using the new custom layered image:

   a. Start a debug session as root for a control plane node:

   ```
   $ oc debug node/<node_name>
   ```

   b. Set **/host** as the root directory within the debug shell:

   ```
   sh-4.4# chroot /host
   ```

   c. Run the **rpm-ostree status** command to view that the custom layered image is in use:

   ```
   sh-5.1# rpm-ostree status
   ```

   Example output

   ```
   # ...
   Deployments:
   * ostree-unverified-registry:quay.io/openshift-release-dev/os-
   image@sha256:f636fa5b504e92e6faa22ecd71a60b089dab72200f3d130c68dfec07148d11
   cd
   ```
   1
   ```
                Digest:
   sha256:bcea2546295b2a55e0a9bf6dd4789433a9867e378661093b6fdee0031ed1e8a4
                Version: 416.94.202405141654-0 (2024-05-14T16:58:43Z)
   ```

   1 Digested image pull spec for the new custom layered image.

Additional resources

- [Enabling features using feature gates](#)

7.4. USING OUT-OF-CLUSTER LAYERING TO APPLY A CUSTOM LAYERED IMAGE

You can easily configure Red Hat Enterprise Linux CoreOS (RHCOS) image layering on the nodes in specific machine config pools. The Machine Config Operator (MCO) reboots those nodes with the new custom layered image, overriding the base Red Hat Enterprise Linux CoreOS (RHCOS) image.

To apply a custom layered image to your cluster, you must have the custom layered image in a repository that your cluster can access. Then, create a **MachineConfig** object that points to the custom layered image. You need a separate **MachineConfig** object for each machine config pool that you want to configure.

IMPORTANT

When you configure a custom layered image, OpenShift Container Platform no longer automatically updates any node that uses the custom layered image. You become responsible for manually updating your nodes as appropriate. If you roll back the custom layer, OpenShift Container Platform will again automatically update the node. See the Additional resources section that follows for important information about updating nodes that use a custom layered image.

Prerequisites

- You must create a custom layered image that is based on an OpenShift Container Platform image digest, not a tag.

  NOTE

  You should use the same base RHCOS image that is installed on the rest of your cluster. Use the **oc adm release info --image-for rhel-coreos** command to obtain the base image being used in your cluster.

  For example, the following Containerfile creates a custom layered image from an OpenShift Container Platform 4.16 image and overrides the kernel package with one from CentOS 9 Stream:

  Example Containerfile for a custom layer image

  ```
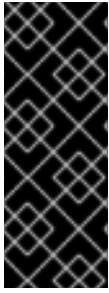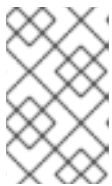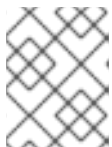  # Using a 4.16.0 image
  FROM quay.io/openshift-release/ocp-release@sha256... 1
  #Install hotfix rpm
  RUN rpm-ostree override replace http://mirror.stream.centos.org/9-
  stream/BaseOS/x86_64/os/Packages/kernel-{,core-,modules-,modules-core-,modules-extra-
  }5.14.0-295.el9.x86_64.rpm && \ 2
      rpm-ostree cleanup -m && \
      ostree container commit
  ```

  1 Specifies the RHCOS base image of your cluster.

  2 Replaces the kernel packages.

  NOTE

  Instructions on how to create a Containerfile are beyond the scope of this documentation.

- Because the process for building a custom layered image is performed outside of the cluster, you must use the **--authfile /path/to/pull-secret** option with Podman or Buildah. Alternatively, to have the pull secret read by these tools automatically, you can add it to one of the default file locations: ~/**.docker/config.json**, **$XDG_RUNTIME_DIR/containers/auth.json**, ~/**.docker/config.json**, or ~/**.dockercfg**. Refer to the **containers-auth.json** man page for more information.

- You must push the custom layered image to a repository that your cluster can access.

Procedure

1. Create a machine config file.

   a. Create a YAML file similar to the following:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: worker ❶
     name: os-layer-custom
   spec:
     osImageURL: quay.io/my-registry/custom-image@sha256... ❷
   ```

   ❶   Specifies the machine config pool to apply the custom layered image.

   ❷   Specifies the path to the custom layered image in the repository.

   b. Create the **MachineConfig** object:

   ```
   $ oc create -f <file_name>.yaml
   ```

   > **IMPORTANT**
   >
   > It is strongly recommended that you test your images outside of your production environment before rolling out to your cluster.

Verification

You can verify that the custom layered image is applied by performing any of the following checks:

1. Check that the worker machine config pool has rolled out with the new machine config:

   a. Check that the new machine config is created:

   ```
   $ oc get mc
   ```

   Sample output

   ```
   NAME                            GENERATEDBYCONTROLLER
   IGNITIONVERSION   AGE
   00-master                       5bdb57489b720096ef912f738b46330a8f577803
   3.2.0          95m
   00-worker                       5bdb57489b720096ef912f738b46330a8f577803
   3.2.0          95m
   01-master-container-runtime
   5bdb57489b720096ef912f738b46330a8f577803   3.2.0           95m
   01-master-kubelet               5bdb57489b720096ef912f738b46330a8f577803
   3.2.0          95m
   01-worker-container-runtime
   5bdb57489b720096ef912f738b46330a8f577803   3.2.0           95m
   01-worker-kubelet               5bdb57489b720096ef912f738b46330a8f577803
   3.2.0          95m
   ```

```
99-master-generated-registries
5bdb57489b720096ef912f738b46330a8f577803   3.2.0          95m
99-master-ssh                                 3.2.0          98m
99-worker-generated-registries
5bdb57489b720096ef912f738b46330a8f577803   3.2.0          95m
99-worker-ssh                                 3.2.0          98m
os-layer-custom                                              10s
rendered-master-15961f1da260f7be141006404d17d39b
5bdb57489b720096ef912f738b46330a8f577803   3.2.0          95m
rendered-worker-5aff604cb1381a4fe07feaf1595a797e
5bdb57489b720096ef912f738b46330a8f577803   3.2.0          95m
rendered-worker-5de4837625b1cbc237de6b22bc0bc873
5bdb57489b720096ef912f738b46330a8f577803   3.2.0          4s
```

1 New machine config

2 New rendered machine config

b. Check that the **osImageURL** value in the new machine config points to the expected image:

```
$ oc describe mc rendered-master-4e8be63aef68b843b546827b6ebe0913
```

Example output

```
Name:        rendered-master-4e8be63aef68b843b546827b6ebe0913
Namespace:
Labels:      <none>
Annotations: machineconfiguration.openshift.io/generated-by-controller-version:
8276d9c1f574481043d3661a1ace1f36cd8c3b62
             machineconfiguration.openshift.io/release-image-version: 4.16.0-ec.3
API Version:  machineconfiguration.openshift.io/v1
Kind:        MachineConfig
...
  Os Image URL: quay.io/my-registry/custom-image@sha256...
```

c. Check that the associated machine config pool is updating with the new machine config:

```
$ oc get mcp
```

Sample output

```
NAME    CONFIG                                 UPDATED  UPDATING  DEGRADED
MACHINECOUNT  READYMACHINECOUNT  UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT  AGE
master  rendered-master-6faecdfa1b25c114a58cf178fbaa45e2  True    False    False
3        3          3              0              39m
worker  rendered-worker-6b000dbc31aaee63c6a2d56d04cd4c1b  False   True
False  3        0          0              0              39m
```

1 When the **UPDATING** field is **True**, the machine config pool is updating with the new machine config. When the field becomes **False**, the worker machine config pool has

d. Check the nodes to see that scheduling on the nodes is disabled. This indicates that the change is being applied:

```
$ oc get nodes
```

Example output

```
NAME                                  STATUS               ROLES             AGE
VERSION
ip-10-0-148-79.us-west-1.compute.internal    Ready              worker            32m
v1.29.4
ip-10-0-155-125.us-west-1.compute.internal   Ready,SchedulingDisabled   worker
35m   v1.29.4
ip-10-0-170-47.us-west-1.compute.internal    Ready              control-plane,master
42m   v1.29.4
ip-10-0-174-77.us-west-1.compute.internal    Ready              control-plane,master
42m   v1.29.4
ip-10-0-211-49.us-west-1.compute.internal    Ready              control-plane,master
42m   v1.29.4
ip-10-0-218-151.us-west-1.compute.internal   Ready               worker            31m
v1.29.4
```

2. When the node is back in the **Ready** state, check that the node is using the custom layered image:

a. Open an **oc debug** session to the node. For example:

```
$ oc debug node/ip-10-0-155-125.us-west-1.compute.internal
```

b. Set **/host** as the root directory within the debug shell:

```
sh-4.4# chroot /host
```

c. Run the **rpm-ostree status** command to view that the custom layered image is in use:

```
sh-4.4# sudo rpm-ostree status
```

Example output

```
State: idle
Deployments:
* ostree-unverified-registry:quay.io/my-registry/...
            Digest: sha256:...
```

Additional resources

Updating with a RHCOS custom layered image

7.5. REMOVING A RHCOS CUSTOM LAYERED IMAGE

You can easily revert Red Hat Enterprise Linux CoreOS (RHCOS) image layering from the nodes in specific machine config pools. The Machine Config Operator (MCO) reboots those nodes with the cluster base Red Hat Enterprise Linux CoreOS (RHCOS) image, overriding the custom layered image.

To remove a Red Hat Enterprise Linux CoreOS (RHCOS) custom layered image from your cluster, you need to delete the machine config that applied the image.

Procedure

1. Delete the machine config that applied the custom layered image.

   ```
   $ oc delete mc os-layer-custom
   ```

   After deleting the machine config, the nodes reboot.

Verification

You can verify that the custom layered image is removed by performing any of the following checks:

1. Check that the worker machine config pool is updating with the previous machine config:

   ```
   $ oc get mcp
   ```

   Sample output

   ```
   NAME    CONFIG                                      UPDATED   UPDATING   DEGRADED
   MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
   DEGRADEDMACHINECOUNT   AGE
   master   rendered-master-6faecdfa1b25c114a58cf178fbaa45e2   True      False      False
   3        3        3        0            39m
   worker   rendered-worker-6b000dbc31aaee63c6a2d56d04cd4c1b   False     True       False
   3        0        0        0            39m ❶
   ```

   ❶ When the **UPDATING** field is **True**, the machine config pool is updating with the previous machine config. When the field becomes **False**, the worker machine config pool has rolled out to the previous machine config.

2. Check the nodes to see that scheduling on the nodes is disabled. This indicates that the change is being applied:

   ```
   $ oc get nodes
   ```

   Example output

   ```
   NAME                               STATUS              ROLES              AGE   VERSION
   ip-10-0-148-79.us-west-1.compute.internal    Ready               worker               32m
   v1.29.4
   ip-10-0-155-125.us-west-1.compute.internal   Ready,SchedulingDisabled   worker
   35m   v1.29.4
   ip-10-0-170-47.us-west-1.compute.internal    Ready               control-plane,master   42m
   v1.29.4
   ip-10-0-174-77.us-west-1.compute.internal    Ready               control-plane,master   42m
   v1.29.4
   ```

```
ip-10-0-211-49.us-west-1.compute.internal     Ready          control-plane,master  42m
v1.29.4
ip-10-0-218-151.us-west-1.compute.internal   Ready          worker                31m
v1.29.4
```

3. When the node is back in the **Ready** state, check that the node is using the base image:

   a. Open an **oc debug** session to the node. For example:

   ```
   $ oc debug node/ip-10-0-155-125.us-west-1.compute.internal
   ```

   b. Set **/host** as the root directory within the debug shell:

   ```
   sh-4.4# chroot /host
   ```

   c. Run the **rpm-ostree status** command to view that the custom layered image is in use:

   ```
   sh-4.4# sudo rpm-ostree status
   ```

   Example output

   ```
   State: idle
   Deployments:
   * ostree-unverified-registry:podman pull quay.io/openshift-release-dev/ocp-
   release@sha256:e2044c3cfebe0ff3a99fc207ac5efe6e07878ad59fd4ad5e41f88cb016dacd
   73
             Digest:
   sha256:e2044c3cfebe0ff3a99fc207ac5efe6e07878ad59fd4ad5e41f88cb016dacd73
   ```

7.6. UPDATING WITH A RHCOS CUSTOM LAYERED IMAGE

When you configure Red Hat Enterprise Linux CoreOS (RHCOS) image layering, OpenShift Container Platform no longer automatically updates the node pool that uses the custom layered image. You become responsible to manually update your nodes as appropriate.

To update a node that uses a custom layered image, follow these general steps:

1. The cluster automatically upgrades to version x.y.z+1, except for the nodes that use the custom layered image.

2. You could then create a new Containerfile that references the updated OpenShift Container Platform image and the RPM that you had previously applied.

3. Create a new machine config that points to the updated custom layered image.

Updating a node with a custom layered image is not required. However, if that node gets too far behind the current OpenShift Container Platform version, you could experience unexpected results.

CHAPTER 8. MACHINE CONFIG DAEMON METRICS OVERVIEW

The Machine Config Daemon is a part of the Machine Config Operator. It runs on every node in the cluster. The Machine Config Daemon manages configuration changes and updates on each of the nodes.

8.1. UNDERSTANDING MACHINE CONFIG DAEMON METRICS

Beginning with OpenShift Container Platform 4.3, the Machine Config Daemon provides a set of metrics. These metrics can be accessed using the Prometheus Cluster Monitoring stack.

The following table describes this set of metrics. Some entries contain commands for getting specific logs. Hpwever, the most comprehensive set of logs is available using the **oc adm must-gather** command.

> **NOTE**
>
> Metrics marked with ***** in the **Name** and **Description** columns represent serious errors that might cause performance problems. Such problems might prevent updates and upgrades from proceeding.

Table 8.1. MCO metrics

Name	Format	Description	Notes
mcd_host_os_and_version	**[]string{"os", "version"}**	Shows the OS that MCD is running on, such as RHCOS or RHEL. In case of RHCOS, the version is provided.	
mcd_drain_err*		Logs errors received during failed drain. *	While drains might need multiple tries to succeed, terminal failed drains prevent updates from proceeding. The **drain_time** metric, which shows how much time the drain took, might help with troubleshooting. For further investigation, see the logs by running: **$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon**

Name	Format	Description	Notes
mcd_pivot_err*	**[]string{"err", "node", "pivot_target"}**	Logs errors encountered during pivot. *	Pivot errors might prevent OS upgrades from proceeding. For further investigation, run this command to see the logs from the **machine-config-daemon** container: **$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon**
mcd_state	**[]string{"state", "reason"}**	State of Machine Config Daemon for the indicated node. Possible states are "Done", "Working", and "Degraded". In case of "Degraded", the reason is included.	For further investigation, see the logs by running: **$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon**
mcd_kubelet_state*		Logs kubelet health failures. *	This is expected to be empty, with failure count of 0. If failure count exceeds 2, the error indicating threshold is exceeded. This indicates a possible issue with the health of the kubelet. For further investigation, run this command to access the node and see all its logs: **$ oc debug node/<node> — chroot /host journalctl -u kubelet**
mcd_reboot_err*	**[]string{"message", "err", "node"}**	Logs the failed reboots and the corresponding errors. *	This is expected to be empty, which indicates a successful reboot. For further investigation, see the logs by running: **$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon**

Name	Format	Description	Notes
mcd_update _state	**[]string{"con fig", "err"}**	Logs success or failure of configuration updates and the corresponding errors.	The expected value is **rendered-master/rendered-worker-XXXX**. If the update fails, an error is present. For further investigation, see the logs by running: **$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-\<hash\> -c machine-config-daemon**

Additional resources

- [Monitoring overview](#)

- [Gathering data about your cluster](#)