OpenShift Container Platform 4.12

Post-installation configuration

Day 2 operations for OpenShift Container Platform

OpenShift Container Platform 4.12 Post-installation configuration

Day 2 operations for OpenShift Container Platform

Legal Notice

Abstract

This document provides instructions and guidance on post installation activities for OpenShift Container Platform.

Table of Contents

CHAPTER 1. POSTINSTALLATION CONFIGURATION OVERVIEW

After installing OpenShift Container Platform, a cluster administrator can configure and customize the following components:

- Machine

- Bare metal

- Cluster

- Node

- Network

- Storage

- Users

- Alerts and notifications

1.1. POST-INSTALLATION CONFIGURATION TASKS

You can perform the post-installation configuration tasks to configure your environment to meet your need.

The following lists details these configurations:

- Configure operating system features: The Machine Config Operator (MCO) manages **MachineConfig** objects. By using the MCO, you can configure nodes and custom resources.

- Configure bare metal nodes: You can use the Bare Metal Operator (BMO) to manage bare metal hosts. The BMO can complete the following operations:

  - Inspects hardware details of the host and report them to the bare metal host.

  - Inspect firmware and configure BIOS settings.

  - Provision hosts with a desired image.

  - Clean disk contents for the host before or after provisioning the host.

- Configure cluster features. You can modify the following features of an OpenShift Container Platform cluster:

  - Image registry

  - Networking configuration

  - Image build behavior

  - Identity provider

  - The etcd configuration

- Machine set creation to handle the workloads

- Cloud provider credential management

- **Configuring a private cluster**: By default, the installation program provisions OpenShift Container Platform by using a publicly accessible DNS and endpoints. To make your cluster accessible only from within an internal network, configure the following components to make them private:

  - DNS

  - Ingress Controller

  - API server

- **Perform node operations**: By default, OpenShift Container Platform uses Red Hat Enterprise Linux CoreOS (RHCOS) compute machines. You can perform the following node operations:

  - Add and remove compute machines.

  - Add and remove taints and tolerations.

  - Configure the maximum number of pods per node.

  - Enable Device Manager.

- **Configure network**: After installing OpenShift Container Platform, you can configure the following components:

  - Ingress cluster traffic

  - Node port service range

  - Network policy

  - Enabling the cluster-wide proxy

- **Configure storage**: By default, containers operate by using the ephemeral storage or transient local storage. The ephemeral storage has a lifetime limitation. To store the data for a long time, you must configure persistent storage. You can configure storage by using one of the following methods:

  - **Dynamic provisioning**: You can dynamically provision storage on demand by defining and creating storage classes that control different levels of storage, including storage access.

  - **Static provisioning**: You can use Kubernetes persistent volumes to make existing storage available to a cluster. Static provisioning can support various device configurations and mount options.

- **Configure users**: OAuth access tokens allow users to authenticate themselves to the API. You can configure OAuth to perform the following tasks:

- Specify an identity provider

- Use role-based access control to define and supply permissions to users

- Install an Operator from OperatorHub

- Configuring alert notifications: By default, firing alerts are displayed on the Alerting UI of the web console. You can also configure OpenShift Container Platform to send alert notifications to external systems.

CHAPTER 2. CONFIGURING A PRIVATE CLUSTER

After you install an OpenShift Container Platform version 4.12 cluster, you can set some of its core components to be private.

2.1. ABOUT PRIVATE CLUSTERS

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

> **IMPORTANT**
>
> If the cluster has any public subnets, load balancer services created by administrators might be publicly accessible. To ensure cluster security, verify that these services are explicitly annotated as private.

DNS

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

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

Ingress Controller

Because the default **Ingress** object is created as public, the load balancer is internet-facing and in the public subnets.

The Ingress Operator generates a default certificate for an Ingress Controller to serve as a placeholder until you configure a custom default certificate. Do not use Operator-generated default certificates in production clusters. The Ingress Operator does not rotate its own signing certificate or the default certificates that it generates. Operator-generated default certificates are intended as placeholders for custom default certificates that you configure.

API server

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

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

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

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

2.2. SETTING DNS TO PRIVATE

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

Procedure

1. Review the **DNS** custom resource for your cluster:

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

   Example output

   ```
   apiVersion: config.openshift.io/v1
   kind: DNS
   metadata:
     creationTimestamp: "2019-10-25T18:27:09Z"
     generation: 2
     name: cluster
     resourceVersion: "37966"
     selfLink: /apis/config.openshift.io/v1/dnses/cluster
     uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
   spec:
     baseDomain: <base_domain>
     privateZone:
       tags:
         Name: <infrastructure_id>-int
         kubernetes.io/cluster/<infrastructure_id>: owned
     publicZone:
       id: Z2XXXXXXXXXXA4
   status: {}
   ```

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

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

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

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

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

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

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

   Example output

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

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

2.3. SETTING THE INGRESS CONTROLLER TO PRIVATE

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

Procedure

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

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

   Example output

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

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

2.4. RESTRICTING THE API SERVER TO PRIVATE

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

Prerequisites

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

- Have access to the web console as a user with **admin** privileges.

Procedure

1. In the web portal or console for your cloud provider, take the following actions:

   a. Locate and delete the appropriate load balancer component:

      - For AWS, delete the external load balancer. The API DNS entry in the private zone already points to the internal load balancer, which uses an identical configuration, so you do not need to modify the internal load balancer.

      - For Azure, delete the **api-internal** rule for the load balancer.

   b. Delete the **api.$clustername.$yourdomain** DNS entry in the public zone.

2. Remove the external load balancers:

   > **IMPORTANT**
   >
   > You can run the following steps only for an installer-provisioned infrastructure (IPI) cluster. For a user-provisioned infrastructure (UPI) cluster, you must manually remove or disable the external load balancers.

   - If your cluster uses a control plane machine set, delete the following lines in the control plane machine set custom resource:

     ```
     providerSpec:
       value:
         loadBalancers:
         - name: lk4pj-ext  ❶
           type: network  ❷
         - name: lk4pj-int
           type: network
     ```

     ❶ ❷ Delete this line.

   - If your cluster does not use a control plane machine set, you must delete the external load balancers from each control plane machine.

     i. From your terminal, list the cluster machines by running the following command:

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

        Example output

        ```
        NAME                      STATE    TYPE       REGION     ZONE        AGE
        lk4pj-master-0            running  m4.xlarge  us-east-1  us-east-1a  17m
        lk4pj-master-1            running  m4.xlarge  us-east-1  us-east-1b  17m
        lk4pj-master-2            running  m4.xlarge  us-east-1  us-east-1a  17m
        lk4pj-worker-us-east-1a-5fzfj  running  m4.xlarge  us-east-1  us-east-1a  15m
        lk4pj-worker-us-east-1a-vbghs  running  m4.xlarge  us-east-1  us-east-1a  15m
        lk4pj-worker-us-east-1b-zgpzg  running  m4.xlarge  us-east-1  us-east-1b  15m
        ```

The control plane machines contain **master** in the name.

   ii. Remove the external load balancer from each control plane machine:

      A. Edit a control plane machine object to by running the following command:

```
$ oc edit machines -n openshift-machine-api <control_plane_name> ❶
```

❶   Specify the name of the control plane machine object to modify.

      B. Remove the lines that describe the external load balancer, which are marked in the following example:

```
providerSpec:
  value:
    loadBalancers:
    - name: lk4pj-ext ❶
      type: network ❷
    - name: lk4pj-int
      type: network
```

❶ ❷ Delete this line.

      C. Save your changes and exit the object specification.

      D. Repeat this process for each of the control plane machines.

2.4.1. Configuring the Ingress Controller endpoint publishing scope to Internal

When a cluster administrator installs a new cluster without specifying that the cluster is private, the default Ingress Controller is created with a **scope** set to **External**. Cluster administrators can change an **External** scoped Ingress Controller to **Internal**.

Prerequisites

- You installed the **oc** CLI.

Procedure

- To change an **External** scoped Ingress Controller to **Internal**, enter the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --
patch='{"spec":{"endpointPublishingStrategy":{"type":"LoadBalancerService","loadBalancer":
{"scope":"Internal"}}}}'
```

- To check the status of the Ingress Controller, enter the following command:

```
$ oc -n openshift-ingress-operator get ingresscontrollers/default -o yaml
```

   ○ The **Progressing** status condition indicates whether you must take further action. For example, the status condition can indicate that you need to delete the service by entering the following command:

```
$ oc -n openshift-ingress delete services/router-default
```

If you delete the service, the Ingress Operator recreates it as **Internal**.

CHAPTER 3. BARE METAL CONFIGURATION

When deploying OpenShift Container Platform on bare metal hosts, there are times when you need to make changes to the host either before or after provisioning. This can include inspecting the host's hardware, firmware, and firmware details. It can also include formatting disks or changing modifiable firmware settings.

3.1. ABOUT THE BARE METAL OPERATOR

Use the Bare Metal Operator (BMO) to provision, manage, and inspect bare-metal hosts in your cluster.

The BMO uses three resources to complete these tasks:

- **BareMetalHost**

- **HostFirmwareSettings**

- **FirmwareSchema**

The BMO maintains an inventory of the physical hosts in the cluster by mapping each bare-metal host to an instance of the **BareMetalHost** custom resource definition. Each **BareMetalHost** resource features hardware, software, and firmware details. The BMO continually inspects the bare-metal hosts in the cluster to ensure each **BareMetalHost** resource accurately details the components of the corresponding host.

The BMO also uses the **HostFirmwareSettings** resource and the **FirmwareSchema** resource to detail firmware specifications for the bare-metal host.

The BMO interfaces with bare-metal hosts in the cluster by using the Ironic API service. The Ironic service uses the Baseboard Management Controller (BMC) on the host to interface with the machine.

Some common tasks you can complete by using the BMO include the following:

- Provision bare-metal hosts to the cluster with a specific image

- Format a host's disk contents before provisioning or after deprovisioning

- Turn on or off a host

- Change firmware settings

- View the host's hardware details

3.1.1. Bare Metal Operator architecture

The Bare Metal Operator (BMO) uses three resources to provision, manage, and inspect bare-metal hosts in your cluster. The following diagram illustrates the architecture of these resources:
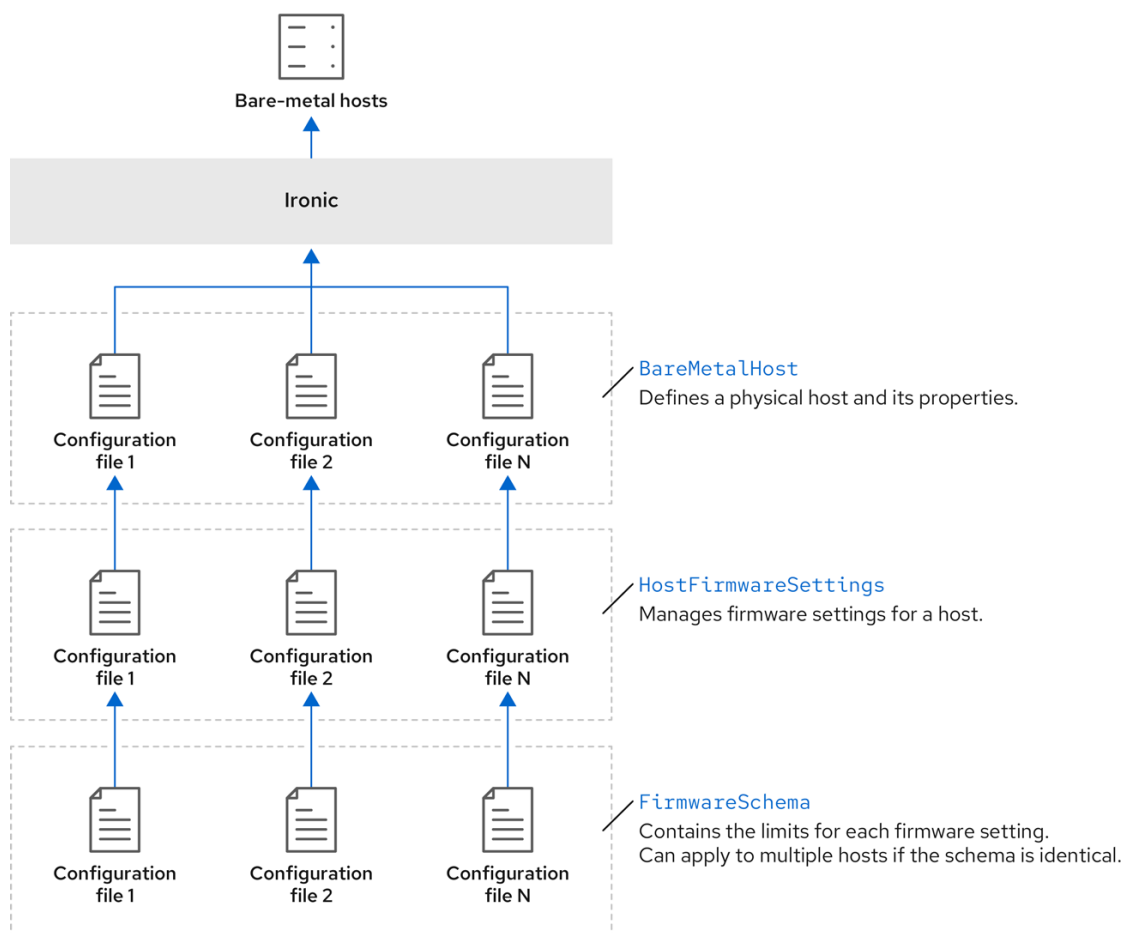
BareMetalHost

The **BareMetalHost** resource defines a physical host and its properties. When you provision a bare-metal host to the cluster, you must define a **BareMetalHost** resource for that host. For ongoing management of the host, you can inspect the information in the **BareMetalHost** or update this information.

The **BareMetalHost** resource features provisioning information such as the following:

- Deployment specifications such as the operating system boot image or the custom RAM disk

- Provisioning state

- Baseboard Management Controller (BMC) address

- Desired power state

The **BareMetalHost** resource features hardware information such as the following:

- Number of CPUs

- MAC address of a NIC

- Size of the host's storage device

- Current power state

HostFirmwareSettings

You can use the **HostFirmwareSettings** resource to retrieve and manage the firmware settings for a host. When a host moves to the **Available** state, the Ironic service reads the host's firmware settings and creates the **HostFirmwareSettings** resource. There is a one-to-one mapping between the **BareMetalHost** resource and the **HostFirmwareSettings** resource.

You can use the **HostFirmwareSettings** resource to inspect the firmware specifications for a host or to update a host's firmware specifications.

> **NOTE**
>
> You must adhere to the schema specific to the vendor firmware when you edit the **spec** field of the **HostFirmwareSettings** resource. This schema is defined in the read-only **FirmwareSchema** resource.

FirmwareSchema

Firmware settings vary among hardware vendors and host models. A **FirmwareSchema** resource is a read-only resource that contains the types and limits for each firmware setting on each host model. The data comes directly from the BMC by using the Ironic service. The **FirmwareSchema** resource enables you to identify valid values you can specify in the **spec** field of the **HostFirmwareSettings** resource.

A **FirmwareSchema** resource can apply to many **BareMetalHost** resources if the schema is the same.

Additional resources

- Metal³ API service for provisioning bare-metal hosts

- Ironic API service for managing bare-metal infrastructure

3.2. ABOUT THE BAREMETALHOST RESOURCE

Metal³ introduces the concept of the **BareMetalHost** resource, which defines a physical host and its properties. The **BareMetalHost** resource contains two sections:

1. The **BareMetalHost** spec

2. The **BareMetalHost** status

3.2.1. The BareMetalHost spec

The **spec** section of the **BareMetalHost** resource defines the desired state of the host.

Table 3.1. BareMetalHost spec

Parameters	Description
automatedCleaningMode	An interface to enable or disable automated cleaning during provisioning and de-provisioning. When set to **disabled**, it skips automated cleaning. When set to **metadata**, automated cleaning is enabled. The default setting is **metadata**.

Parameters	Description

bmc:
  address:
  credentialsName:
  disableCertificateVerification:
	The **bmc** configuration setting contains the connection information for the baseboard management controller (BMC) on the host. The fields are:
	
	• **address**: The URL for communicating with the host's BMC controller.
	
	• **credentialsName**: A reference to a secret containing the username and password for the BMC.
	
	• **disableCertificateVerification**: A boolean to skip certificate validation when set to **true**.

bootMACAddress	The MAC address of the NIC used for provisioning the host.
bootMode	The boot mode of the host. It defaults to **UEFI**, but it can also be set to **legacy** for BIOS boot, or **UEFISecureBoot**.
consumerRef	A reference to another resource that is using the host. It could be empty if another resource is not currently using the host. For example, a **Machine** resource might use the host when the **machine-api** is using the host.
description	A human-provided string to help identify the host.
externallyProvisioned	A boolean indicating whether the host provisioning and deprovisioning are managed externally. When set: • Power status can still be managed using the online field. • Hardware inventory will be monitored, but no provisioning or deprovisioning operations are performed on the host.

Parameters	Description
firmware	Contains information about the BIOS configuration of bare metal hosts. Currently, **firmware** is only supported by iRMC, iDRAC, iLO4 and iLO5 BMCs. The sub fields are: • **simultaneousMultithreadingEnabled**: Allows a single physical processor core to appear as several logical processors. Valid settings are **true** or **false**. • **sriovEnabled**: SR-IOV support enables a hypervisor to create virtual instances of a PCI-express device, potentially increasing performance. Valid settings are **true** or **false**. • **virtualizationEnabled**: Supports the virtualization of platform hardware. Valid settings are **true** or **false**.

image:
  url:
  checksum:
  checksumType:
  format:
	The **image** configuration setting holds the details for the image to be deployed on the host. Ironic requires the image fields. However, when the **externallyProvisioned** configuration setting is set to **true** and the external management doesn't require power control, the fields can be empty. The fields are:
	
	• **url**: The URL of an image to deploy to the host.
	
	• **checksum**: The actual checksum or a URL to a file containing the checksum for the image at **image.url**.
	
	• **checksumType**: You can specify checksum algorithms. Currently **image.checksumType** only supports **md5**, **sha256**, and **sha512**. The default checksum type is **md5**.
	
	• **format**: This is the disk format of the image. It can be one of **raw**, **qcow2**, **vdi**, **vmdk**, **live-iso** or be left unset. Setting it to **raw** enables raw image streaming in the Ironic agent for that image. Setting it to **live-iso** enables iso images to live boot without deploying to disk, and it ignores the **checksum** fields.

Parameters	Description
networkData	A reference to the secret containing the network configuration data and its namespace, so that it can be attached to the host before the host boots to set up the network.
online	A boolean indicating whether the host should be powered on (**true**) or off (**false**). Changing this value will trigger a change in the power state of the physical host.

raid:
    hardwareRAIDVolumes:
    softwareRAIDVolumes:
	(Optional) Contains the information about the RAID configuration for bare metal hosts. If not specified, it retains the current configuration.
	
	NOTE
	
	OpenShift Container Platform 4.12 supports hardware RAID for BMCs using the iRMC protocol only. OpenShift Container Platform 4.12 does not support software RAID.
	
	See the following configuration settings:
	
	• **hardwareRAIDVolumes**: Contains the list of logical drives for hardware RAID, and defines the desired volume configuration in the hardware RAID. If you don't specify **rootDeviceHints**, the first volume is the root volume. The sub-fields are:
	
	  ○ **level**: The RAID level for the logical drive. The following levels are supported: **0**,**1**,**2**,**5**,**6**,**1+0**,**5+0**,**6+0**.
	
	  ○ **name**: The name of the volume as a string. It should be unique within the server. If not specified, the volume name will be auto-generated.
	
	  ○ **numberOfPhysicalDisks**: The number of physical drives as an integer to use for the logical drove. Defaults to the minimum number of disk drives required for the particular RAID level.
	
	  ○ **physicalDisks**: The list of names of physical disk drives as a string. This is an optional field. If specified, the controller field must be specified too.
	
	  ○ **controller**: (Optional) The name of the RAID controller as a string to use in the hardware RAID volume.
	
	  ○ **rotational**: If set to **true**, it will only select rotational disk drives. If set to

Parameters	Description
	false, it will only select solid-state and NVMe drives. If not set, it selects any drive types, which is the default behavior. ○ **sizeGibibytes**: The size of the logical drive as an integer to create in GiB. If unspecified or set to **0**, it will use the maximum capacity of physical drive for the logical drive. • **softwareRAIDVolumes**: OpenShift Container Platform 4.12 does not support software RAID. The following information is for reference only. This configuration contains the list of logical disks for software RAID. If you don't specify **rootDeviceHints**, the first volume is the root volume. If you set **HardwareRAIDVolumes**, this item will be invalid. Software RAIDs will always be deleted. The number of created software RAID devices must be **1** or **2**. If there is only one software RAID device, it must be **RAID-1**. If there are two RAID devices, the first device must be **RAID-1**, while the RAID level for the second device can be **0**, **1**, or **1+0**. The first RAID device will be the deployment device. Therefore, enforcing **RAID-1** reduces the risk of a non-booting node in case of a device failure. The **softwareRAIDVolume** field defines the desired configuration of the volume in the software RAID. The sub-fields are: ○ **level**: The RAID level for the logical drive. The following levels are supported: **0**,**1**,**1+0**. ○ **physicalDisks**: A list of device hints. The number of items should be greater than or equal to **2**. ○ **sizeGibibytes**: The size of the logical disk drive as an integer to be created in GiB. If unspecified or set to **0**, it will use the maximum capacity of physical drive for logical drive. You can set the **hardwareRAIDVolume** as an empty slice to clear the hardware RAID configuration. For example:

```
spec:
  raid:
    hardwareRAIDVolume: []
```

If you receive an error message indicating that the driver does not support RAID, set the **raid**, **hardwareRAIDVolumes** or **softwareRAIDVolumes** to nil. You might need to ensure the host has a RAID controller.

Parameters	Description

rootDeviceHints:
  deviceName:
  hctl:
  model:
  vendor:
  serialNumber:
  minSizeGigabytes:
  wwn:
  wwnWithExtension:
  wwnVendorExtension:
  rotational:
	The **rootDeviceHints** parameter enables provisioning of the RHCOS image to a particular device. It examines the devices in the order it discovers them, and compares the discovered values with the hint values. It uses the first discovered device that matches the hint value. The configuration can combine multiple hints, but a device must match all hints to get selected. The fields are:
	
	• **deviceName**: A string containing a Linux device name like **/dev/vda**. The hint must match the actual value exactly.
	
	• **hctl**: A string containing a SCSI bus address like **0:0:0:0**. The hint must match the actual value exactly.
	
	• **model**: A string containing a vendor-specific device identifier. The hint can be a substring of the actual value.
	
	• **vendor**: A string containing the name of the vendor or manufacturer of the device. The hint can be a sub-string of the actual value.
	
	• **serialNumber**: A string containing the device serial number. The hint must match the actual value exactly.
	
	• **minSizeGigabytes**: An integer representing the minimum size of the device in gigabytes.
	
	• **wwn**: A string containing the unique storage identifier. The hint must match the actual value exactly.
	
	• **wwnWithExtension**: A string containing the unique storage identifier with the vendor extension appended. The hint must match the actual value exactly.
	
	• **wwnVendorExtension**: A string containing the unique vendor storage identifier. The hint must match the actual value exactly.
	
	• **rotational**: A boolean indicating whether the device should be a rotating disk (true) or not (false).

3.2.2. The BareMetalHost status

The **BareMetalHost** status represents the host's current state, and includes tested credentials, current hardware details, and other information.

Table 3.2. BareMetalHost status

Parameters	Description
goodCredentials	A reference to the secret and its namespace holding the last set of baseboard management controller (BMC) credentials the system was able to validate as working.
errorMessage	Details of the last error reported by the provisioning backend, if any.
errorType	Indicates the class of problem that has caused the host to enter an error state. The error types are: • **provisioned registration error**: Occurs when the controller is unable to re-register an already provisioned host. • **registration error**: Occurs when the controller is unable to connect to the host's baseboard management controller. • **inspection error**: Occurs when an attempt to obtain hardware details from the host fails. • **preparation error**: Occurs when cleaning fails. • **provisioning error**: Occurs when the controller fails to provision or deprovision the host. • **power management error**: Occurs when the controller is unable to modify the power state of the host. • **detach error**: Occurs when the controller is unable to detatch the host from the provisioner.

`hardware:`
`  cpu`
`    arch:`
`    model:`
`    clockMegahertz:`
`    flags:`
`    count:`
	The **hardware.cpu** field details of the CPU(s) in the system. The fields include:
	
	• **arch**: The architecture of the CPU.
	
	• **model**: The CPU model as a string.
	
	• **clockMegahertz**: The speed in MHz of the CPU.
	
	• **flags**: The list of CPU flags. For example, **'mmx','sse','sse2','vmx'** etc.
	
	• **count**: The number of CPUs available in the system.

Parameters	Description
hardware: firmware:	Contains BIOS firmware information. For example, the hardware vendor and version.

hardware:
  nics:
  - ip:
    name:
    mac:
    speedGbps:
    vlans:
    vlanId:
    pxe:
	The **hardware.nics** field contains a list of network interfaces for the host. The fields include:
	
	• **ip**: The IP address of the NIC, if one was assigned when the discovery agent ran.
	
	• **name**: A string identifying the network device. For example, **nic-1**.
	
	• **mac**: The MAC address of the NIC.
	
	• **speedGbps**: The speed of the device in Gbps.
	
	• **vlans**: A list holding all the VLANs available for this NIC.
	
	• **vlanId**: The untagged VLAN ID.
	
	• **pxe**: Whether the NIC is able to boot using PXE.

hardware: ramMebibytes:	The host's amount of memory in Mebibytes (MiB).

hardware:
  storage:
  - name:
    rotational:
    sizeBytes:
    serialNumber:
	The **hardware.storage** field contains a list of storage devices available to the host. The fields include:
	
	• **name**: A string identifying the storage device. For example, **disk 1 (boot)**.
	
	• **rotational**: Indicates whether the disk is rotational, and returns either **true** or **false**.
	
	• **sizeBytes**: The size of the storage device.
	
	• **serialNumber**: The device's serial number.

hardware:
  systemVendor:
    manufacturer:
    productName:
    serialNumber:
	Contains information about the host's **manufacturer**, the **productName**, and the **serialNumber**.

Parameters	Description
lastUpdated	The timestamp of the last time the status of the host was updated.
operationalStatus	The status of the server. The status is one of the following: • **OK**: Indicates all the details for the host are known, correctly configured, working, and manageable. • **discovered**: Implies some of the host's details are either not working correctly or missing. For example, the BMC address is known but the login credentials are not. • **error**: Indicates the system found some sort of irrecoverable error. Refer to the **errorMessage** field in the status section for more details. • **delayed**: Indicates that provisioning is delayed to limit simultaneous provisioning of multiple hosts. • **detached**: Indicates the host is marked **unmanaged**.
poweredOn	Boolean indicating whether the host is powered on.

<pre>provisioning:
  state:
  id:
  image:
  raid:
  firmware:
  rootDeviceHints:</pre>
	The **provisioning** field contains values related to deploying an image to the host. The sub-fields include:
	
	• **state**: The current state of any ongoing provisioning operation. The states include:
	
	  ○ **<empty string>**: There is no provisioning happening at the moment.
	
	  ○ **unmanaged**: There is insufficient information available to register the host.
	
	  ○ **registering**: The agent is checking the host's BMC details.
	
	  ○ **match profile**: The agent is comparing the discovered hardware details on the host against known profiles.
	
	  ○ **available**: The host is available for provisioning. This state was previously known as **ready**.

Parameters	Description
	○ **preparing**: The existing configuration will be removed, and the new configuration will be set on the host.
	○ **provisioning**: The provisioner is writing an image to the host's storage.
	○ **provisioned**: The provisioner wrote an image to the host's storage.
	○ **externally provisioned**: Metal³ does not manage the image on the host.
	○ **deprovisioning**: The provisioner is wiping the image from the host's storage.
	○ **inspecting**: The agent is collecting hardware details for the host.
	○ **deleting**: The agent is deleting the from the cluster.
	● **id**: The unique identifier for the service in the underlying provisioning tool.
	● **image**: The image most recently provisioned to the host.
	● **raid**: The list of hardware or software RAID volumes recently set.
	● **firmware**: The BIOS configuration for the bare metal server.
	● **rootDeviceHints**: The root device selection instructions used for the most recent provisioning operation.
triedCredentials	A reference to the secret and its namespace holding the last set of BMC credentials that were sent to the provisioning backend.

3.3. GETTING THE BAREMETALHOST RESOURCE

The **BareMetalHost** resource contains the properties of a physical host. You must get the **BareMetalHost** resource for a physical host to review its properties.

Procedure

1. Get the list of **BareMetalHost** resources:

   ```
   $ oc get bmh -n openshift-machine-api -o yaml
   ```
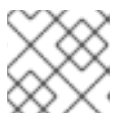
   > **NOTE**
   >
   > You can use **baremetalhost** as the long form of **bmh** with **oc get** command.

2. Get the list of hosts:

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

3. Get the **BareMetalHost** resource for a specific host:

```
$ oc get bmh <host_name> -n openshift-machine-api -o yaml
```

Where **<host_name>** is the name of the host.

Example output

```
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  creationTimestamp: "2022-06-16T10:48:33Z"
  finalizers:
  - baremetalhost.metal3.io
  generation: 2
  name: openshift-worker-0
  namespace: openshift-machine-api
  resourceVersion: "30099"
  uid: 1513ae9b-e092-409d-be1b-ad08edeb1271
spec:
  automatedCleaningMode: metadata
  bmc:
    address: redfish://10.46.61.19:443/redfish/v1/Systems/1
    credentialsName: openshift-worker-0-bmc-secret
    disableCertificateVerification: true
  bootMACAddress: 48:df:37:c7:f7:b0
  bootMode: UEFI
  consumerRef:
    apiVersion: machine.openshift.io/v1beta1
    kind: Machine
    name: ocp-edge-958fk-worker-0-nrfcg
    namespace: openshift-machine-api
  customDeploy:
    method: install_coreos
  online: true
  rootDeviceHints:
    deviceName: /dev/sda
  userData:
    name: worker-user-data-managed
    namespace: openshift-machine-api
status:
  errorCount: 0
  errorMessage: ""
  goodCredentials:
    credentials:
      name: openshift-worker-0-bmc-secret
      namespace: openshift-machine-api
    credentialsVersion: "16120"
  hardware:
    cpu:
      arch: x86_64
```

```
    clockMegahertz: 2300
    count: 64
    flags:
    - 3dnowprefetch
    - abm
    - acpi
    - adx
    - aes
    model: Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz
  firmware:
    bios:
      date: 10/26/2020
      vendor: HPE
      version: U30
  hostname: openshift-worker-0
  nics:
  - mac: 48:df:37:c7:f7:b3
    model: 0x8086 0x1572
    name: ens1f3
  ramMebibytes: 262144
  storage:
  - hctl: "0:0:0:0"
    model: VK000960GWTTB
    name: /dev/sda
    sizeBytes: 960197124096
    type: SSD
    vendor: ATA
  systemVendor:
    manufacturer: HPE
    productName: ProLiant DL380 Gen10 (868703-B21)
    serialNumber: CZ200606M3
lastUpdated: "2022-06-16T11:41:42Z"
operationalStatus: OK
poweredOn: true
provisioning:
  ID: 217baa14-cfcf-4196-b764-744e184a3413
  bootMode: UEFI
  customDeploy:
    method: install_coreos
  image:
    url: ""
  raid:
    hardwareRAIDVolumes: null
    softwareRAIDVolumes: []
  rootDeviceHints:
    deviceName: /dev/sda
  state: provisioned
triedCredentials:
  credentials:
    name: openshift-worker-0-bmc-secret
    namespace: openshift-machine-api
  credentialsVersion: "16120"
```

3.4. ABOUT THE HOSTFIRMWARESETTINGS RESOURCE

You can use the **HostFirmwareSettings** resource to retrieve and manage the BIOS settings for a host. When a host moves to the **Available** state, Ironic reads the host's BIOS settings and creates the **HostFirmwareSettings** resource. The resource contains the complete BIOS configuration returned from the baseboard management controller (BMC). Whereas, the **firmware** field in the **BareMetalHost** resource returns three vendor-independent fields, the **HostFirmwareSettings** resource typically comprises many BIOS settings of vendor-specific fields per host.

The **HostFirmwareSettings** resource contains two sections:

1. The **HostFirmwareSettings** spec.

2. The **HostFirmwareSettings** status.

3.4.1. The `HostFirmwareSettings` spec

The **spec** section of the **HostFirmwareSettings** resource defines the desired state of the host's BIOS, and it is empty by default. Ironic uses the settings in the **spec.settings** section to update the baseboard management controller (BMC) when the host is in the **Preparing** state. Use the **FirmwareSchema** resource to ensure that you do not send invalid name/value pairs to hosts. See "About the FirmwareSchema resource" for additional details.

Example

```
spec:
  settings:
    ProcTurboMode: Disabled
```
1

1　In the foregoing example, the **spec.settings** section contains a name/value pair that will set the **ProcTurboMode** BIOS setting to **Disabled**.

> **NOTE**
>
> Integer parameters listed in the **status** section appear as strings. For example, **"1"**. When setting integers in the **spec.settings** section, the values should be set as integers without quotes. For example, **1**.

3.4.2. The `HostFirmwareSettings` status

The **status** represents the current state of the host's BIOS.

Table 3.3. HostFirmwareSettings

Parameters	Description
| ```
status:
  conditions:
  - lastTransitionTime:
    message:
    observedGeneration:
    reason:
    status:
    type:
``` | The **conditions** field contains a list of state changes. The sub-fields include:<br><br>• **lastTransitionTime**: The last time the state changed.<br><br>• **message**: A description of the state change.<br><br>• **observedGeneration**: The current generation of the **status**. If **metadata.generation** and this field are not the same, the **status.conditions** might be out of date.<br><br>• **reason**: The reason for the state change.<br><br>• **status**: The status of the state change. The status can be **True**, **False** or **Unknown**.<br><br>• **type**: The type of state change. The types are **Valid** and **ChangeDetected**. |
| ```
status:
  schema:
    name:
    namespace:
    lastUpdated:
``` | The **FirmwareSchema** for the firmware settings. The fields include:<br><br>• **name**: The name or unique identifier referencing the schema.<br><br>• **namespace**: The namespace where the schema is stored.<br><br>• **lastUpdated**: The last time the resource was updated. |
| ```
status:
  settings:
``` | The **settings** field contains a list of name/value pairs of a host's current BIOS settings. |

3.5. GETTING THE HOSTFIRMWARESETTINGS RESOURCE

The **HostFirmwareSettings** resource contains the vendor-specific BIOS properties of a physical host. You must get the **HostFirmwareSettings** resource for a physical host to review its BIOS properties.

Procedure

1. Get the detailed list of **HostFirmwareSettings** resources:

   ```
   $ oc get hfs -n openshift-machine-api -o yaml
   ```

> **NOTE**
>
> You can use **hostfirmwaresettings** as the long form of **hfs** with the **oc get** command.

2. Get the list of **HostFirmwareSettings** resources:

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

3. Get the **HostFirmwareSettings** resource for a particular host

```
$ oc get hfs <host_name> -n openshift-machine-api -o yaml
```

Where **<host_name>** is the name of the host.

3.6. EDITING THE HOSTFIRMWARESETTINGS RESOURCE

You can edit the **HostFirmwareSettings** of provisioned hosts.

> **IMPORTANT**
>
> You can only edit hosts when they are in the **provisioned** state, excluding read-only values. You cannot edit hosts in the **externally provisioned** state.

Procedure

1. Get the list of **HostFirmwareSettings** resources:

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

2. Edit a host's **HostFirmwareSettings** resource:

```
$ oc edit hfs <host_name> -n openshift-machine-api
```

Where **<host_name>** is the name of a provisioned host. The **HostFirmwareSettings** resource will open in the default editor for your terminal.

3. Add name/value pairs to the **spec.settings** section:

Example

```
spec:
  settings:
    name: value
```
1

1 Use the **FirmwareSchema** resource to identify the available settings for the host. You cannot set values that are read-only.

4. Save the changes and exit the editor.

5. Get the host's machine name:

```
$ oc get bmh <host_name> -n openshift-machine name
```

Where **<host_name>** is the name of the host. The machine name appears under the **CONSUMER** field.

6. Annotate the machine to delete it from the machineset:

```
$ oc annotate machine <machine_name> machine.openshift.io/delete-machine=true -n openshift-machine-api
```

Where **<machine_name>** is the name of the machine to delete.

7. Get a list of nodes and count the number of worker nodes:

```
$ oc get nodes
```

8. Get the machineset:

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

9. Scale the machineset:

```
$ oc scale machineset <machineset_name> -n openshift-machine-api --replicas=<n-1>
```

Where **<machineset_name>** is the name of the machineset and **<n-1>** is the decremented number of worker nodes.

10. When the host enters the **Available** state, scale up the machineset to make the **HostFirmwareSettings** resource changes take effect:

```
$ oc scale machineset <machineset_name> -n openshift-machine-api --replicas=<n>
```

Where **<machineset_name>** is the name of the machineset and **<n>** is the number of worker nodes.

3.7. VERIFYING THE HOSTFIRMWARE SETTINGS RESOURCE IS VALID

When the user edits the **spec.settings** section to make a change to the **HostFirmwareSetting**(HFS) resource, the Bare Metal Operator (BMO) validates the change against the **FimwareSchema** resource, which is a read-only resource. If the setting is invalid, the BMO will set the **Type** value of the **status.Condition** setting to **False** and also generate an event and store it in the HFS resource. Use the following procedure to verify that the resource is valid.

Procedure

1. Get a list of **HostFirmwareSetting** resources:

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

2. Verify that the **HostFirmwareSettings** resource for a particular host is valid:

```
$ oc describe hfs <host_name> -n openshift-machine-api
```

Where **<host_name>** is the name of the host.

Example output

```
Events:
 Type    Reason          Age    From                                  Message
 ----    ------          ----   ----                                  -------
  Normal  ValidationFailed 2m49s  metal3-hostfirmwaresettings-controller  Invalid BIOS
setting: Setting ProcTurboMode is invalid, unknown enumeration value - Foo
```

IMPORTANT

If the response returns **ValidationFailed**, there is an error in the resource configuration and you must update the values to conform to the **FirmwareSchema** resource.

3.8. ABOUT THE FIRMWARESCHEMA RESOURCE

BIOS settings vary among hardware vendors and host models. A **FirmwareSchema** resource is a read-only resource that contains the types and limits for each BIOS setting on each host model. The data comes directly from the BMC through Ironic. The **FirmwareSchema** enables you to identify valid values you can specify in the **spec** field of the **HostFirmwareSettings** resource. The **FirmwareSchema** resource has a unique identifier derived from its settings and limits. Identical host models use the same **FirmwareSchema** identifier. It is likely that multiple instances of **HostFirmwareSettings** use the same **FirmwareSchema**.

Table 3.4. FirmwareSchema specification

Parameters	Description

Parameters	Description

<BIOS_setting_name>
  attribute_type:
  allowable_values:
  lower_bound:
  upper_bound:
  min_length:
  max_length:
  read_only:
  unique:
	The **spec** is a simple map consisting of the BIOS setting name and the limits of the setting. The fields include:
	
	• **attribute_type**: The type of setting. The supported types are:
	
	   ○ **Enumeration**
	
	   ○ **Integer**
	
	   ○ **String**
	
	   ○ **Boolean**
	
	• **allowable_values**: A list of allowable values when the **attribute_type** is **Enumeration**.
	
	• **lower_bound**: The lowest allowed value when **attribute_type** is **Integer**.
	
	• **upper_bound**: The highest allowed value when **attribute_type** is **Integer**.
	
	• **min_length**: The shortest string length that the value can have when **attribute_type** is **String**.
	
	• **max_length**: The longest string length that the value can have when **attribute_type** is **String**.
	
	• **read_only**: The setting is read only and cannot be modified.
	
	• **unique**: The setting is specific to this host.

3.9. GETTING THE FIRMWARESCHEMA RESOURCE

Each host model from each vendor has different BIOS settings. When editing the **HostFirmwareSettings** resource's **spec** section, the name/value pairs you set must conform to that host's firmware schema. To ensure you are setting valid name/value pairs, get the **FirmwareSchema** for the host and review it.

Procedure

1. To get a list of **FirmwareSchema** resource instances, execute the following:

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

2. To get a particular **FirmwareSchema** instance, execute:

   ```
   $ oc get firmwareschema <instance_name> -n openshift-machine-api -o yaml
   ```

Where **\<instance_name\>** is the name of the schema instance stated in the **HostFirmwareSettings** resource (see Table 3).

CHAPTER 4. CONFIGURING MULTI-ARCHITECTURE COMPUTE MACHINES ON AN OPENSHIFT CONTAINER PLATFORM CLUSTER

An OpenShift Container Platform cluster with multi-architecture compute machines is a cluster that supports compute machines with different architectures. You can deploy a cluster with multi-architecture compute machines by creating an Azure installer-provisioned cluster using the multi-architecture installer binary. For Azure installation, see Installing a cluster on Azure with customizations .

> ### WARNING
>
> The multi-architecture compute machines Technology Preview feature has limited usability with installing, upgrading, and running payloads.

The following procedures explain how to generate an ARM64 boot image and create an Azure compute machine set with the ARM64 boot image. This adds ARM64 compute nodes to your cluster and deploys the desired amount of ARM64 virtual machines (VM). This section also shows how to upgrade your existing cluster to a cluster that supports multi-architecture compute machines. Clusters with multi-architecture compute machines are only available on Azure installer-provisioned infrastructures with x86_64 control plane machines.

> ### IMPORTANT
>
> OpenShift Container Platform clusters with multi-architecture compute machines on Azure installer-provisioned infrastructure installations 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 .

4.1. CREATING AN ARM64 BOOT IMAGE USING THE AZURE IMAGE GALLERY

To configure your cluster with multi-architecture compute machines, you must create an ARM64 boot image and add it to your Azure compute machine set. The following procedure describes how to manually generate an ARM64 boot image.

Prerequisites

- You installed the Azure CLI (**az**).

- You created a single-architecture Azure installer-provisioned cluster with the multi-architecture installer binary.

Procedure

1. Log in to your Azure account:

   ```
   $ az login
   ```

2. Create a storage account and upload the ARM64 virtual hard disk (VHD) to your storage account. The OpenShift Container Platform installation program creates a resource group, however, the boot image can also be uploaded to a custom named resource group:

   ```
   $ az storage account create -n ${STORAGE_ACCOUNT_NAME} -g
   ${RESOURCE_GROUP} -l westus --sku Standard_LRS ❶
   ```

   ❶ The **westus** object is an example region.

3. Create a storage container using the storage account you generated:

   ```
   $ az storage container create -n ${CONTAINER_NAME} --account-name
   ${STORAGE_ACCOUNT_NAME}
   ```

4. You must use the OpenShift Container Platform installation program JSON file to extract the URL and **aarch64** VHD name:

   a. Extract the **URL** field and set it to **RHCOS_VHD_ORIGIN_URL** as the file name by running the following command:

   ```
   $ RHCOS_VHD_ORIGIN_URL=$(oc -n openshift-machine-config-operator get
   configmap/coreos-bootimages -o jsonpath='{.data.stream}' | jq -r
   '.architectures.aarch64."rhel-coreos-extensions"."azure-disk".url')
   ```

   b. Extract the **aarch64** VHD name and set it to **BLOB_NAME** as the file name by running the following command:

   ```
   $ BLOB_NAME=rhcos-$(oc -n openshift-machine-config-operator get configmap/coreos-
   bootimages -o jsonpath='{.data.stream}' | jq -r '.architectures.aarch64."rhel-coreos-
   extensions"."azure-disk".release')-azure.aarch64.vhd
   ```

5. Generate a shared access signature (SAS) token. Use this token to upload the RHCOS VHD to your storage container with the following commands:

   ```
   $ end=`date -u -d "30 minutes" '+%Y-%m-%dT%H:%MZ'`
   ```

   ```
   $ sas=`az storage container generate-sas -n ${CONTAINER_NAME} --account-name
   ${STORAGE_ACCOUNT_NAME} --https-only --permissions dlrw --expiry $end -o tsv`
   ```

6. Copy the RHCOS VHD into the storage container:

   ```
   $ az storage blob copy start --account-name ${STORAGE_ACCOUNT_NAME} --sas-token
   "$sas" \
     --source-uri "${RHCOS_VHD_ORIGIN_URL}" \
     --destination-blob "${BLOB_NAME}" --destination-container ${CONTAINER_NAME}
   ```

You can check the status of the copying process with the following command:

```
$ az storage blob show -c ${CONTAINER_NAME} -n ${BLOB_NAME} --account-name
${STORAGE_ACCOUNT_NAME} | jq .properties.copy
```

Example output

```
{
 "completionTime": null,
 "destinationSnapshot": null,
 "id": "1fd97630-03ca-489a-8c4e-cfe839c9627d",
 "incrementalCopy": null,
 "progress": "17179869696/17179869696",
 "source": "https://rhcos.blob.core.windows.net/imagebucket/rhcos-411.86.202207130959-0-
azure.aarch64.vhd",
 "status": "success", ❶
 "statusDescription": null
}
```

❶   If the status parameter displays the **success** object, the copying process is complete.

7. Create an image gallery using the following command:

```
$ az sig create --resource-group ${RESOURCE_GROUP} --gallery-name
${GALLERY_NAME}
```

Use the image gallery to create an image definition. In the following example command, **rhcos-arm64** is the name of the image definition.

```
$ az sig image-definition create --resource-group ${RESOURCE_GROUP} --gallery-name
${GALLERY_NAME} --gallery-image-definition rhcos-arm64 --publisher RedHat --offer arm -
-sku arm64 --os-type linux --architecture Arm64 --hyper-v-generation V2
```

8. To get the URL of the VHD and set it to **RHCOS_VHD_URL** as the file name, run the following command:

```
$ RHCOS_VHD_URL=$(az storage blob url --account-name
${STORAGE_ACCOUNT_NAME} -c ${CONTAINER_NAME} -n "${BLOB_NAME}" -o tsv)
```

9. Use the **RHCOS_VHD_URL** file, your storage account, resource group, and image gallery to create an image version. In the following example, **1.0.0** is the image version.

```
$ az sig image-version create --resource-group ${RESOURCE_GROUP} --gallery-name
${GALLERY_NAME} --gallery-image-definition rhcos-arm64 --gallery-image-version 1.0.0 --
os-vhd-storage-account ${STORAGE_ACCOUNT_NAME} --os-vhd-uri
${RHCOS_VHD_URL}
```

10. Your ARM64 boot image is now generated. You can access the ID of your image with the following command:

```
$ az sig image-version show -r $GALLERY_NAME -g $RESOURCE_GROUP -i rhcos-arm64
-e 1.0.0
```

The following example image ID is used in the **recourseID** parameter of the compute machine set:

Example resourceID

> /resourceGroups/${RESOURCE_GROUP}/providers/Microsoft.Compute/galleries/${GALLERY
> _NAME}/images/rhcos-arm64/versions/1.0.0

4.2. ADDING A MULTI-ARCHITECTURE COMPUTE MACHINE SET TO YOUR CLUSTER USING THE ARM64 BOOT IMAGE

To add ARM64 compute nodes to your cluster, you must create an Azure compute machine set that uses the ARM64 boot image. To create your own custom compute machine set on Azure, see "Creating a compute machine set on Azure".

Prerequisites

- You installed the OpenShift CLI (**oc**).

Procedure

- Create a machine set and modify the **resourceID** and **vmSize** parameters with the following command. This machine set will control the ARM64 worker nodes in your cluster:

> $ oc create -f arm64-machine-set-0.yaml

Sample YAML machine set with ARM64 boot image

+

```yaml
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id>
    machine.openshift.io/cluster-api-machine-role: worker
    machine.openshift.io/cluster-api-machine-type: worker
  name: <infrastructure_id>-arm64-machine-set-0
  namespace: openshift-machine-api
spec:
  replicas: 2
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-arm64-machine-set-0
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-arm64-machine-set-0
    spec:
      lifecycleHooks: {}
```

```
  metadata: {}
  providerSpec:
    value:
      acceleratedNetworking: true
      apiVersion: machine.openshift.io/v1beta1
      credentialsSecret:
        name: azure-cloud-credentials
        namespace: openshift-machine-api
      image:
        offer: ""
        publisher: ""
        resourceID:
/resourceGroups/${RESOURCE_GROUP}/providers/Microsoft.Compute/galleries/${GALLERY_NAME}
/images/rhcos-arm64/versions/1.0.0 ❶
        sku: ""
        version: ""
      kind: AzureMachineProviderSpec
      location: <region>
      managedIdentity: <infrastructure_id>-identity
      networkResourceGroup: <infrastructure_id>-rg
      osDisk:
        diskSettings: {}
        diskSizeGB: 128
        managedDisk:
          storageAccountType: Premium_LRS
        osType: Linux
      publicIP: false
      publicLoadBalancer: <infrastructure_id>
      resourceGroup: <infrastructure_id>-rg
      subnet: <infrastructure_id>-worker-subnet
      userDataSecret:
        name: worker-user-data
      vmSize: Standard_D4ps_v5 ❷
      vnet: <infrastructure_id>-vnet
      zone: "<zone>"
```

❶ Set the **resourceID** parameter to the **arm64** boot image.

❷ Set the **vmSize** parameter to the instance type used in your installation. Some example instance types are **Standard_D4ps_v5** or **D8ps**.

Verification

1. Verify that the new ARM64 machines are running by entering the following command:

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

   Example output

   ```
   NAME                                   DESIRED  CURRENT  READY  AVAILABLE  AGE
   <infrastructure_id>-arm64-machine-set-0        2        2      2          2 10m
   ```

2. You can check that the nodes are ready and schedulable with the following command:

```
$ oc get nodes
```

Additional resources

- [Creating a compute machine set on Azure](#)

4.3. UPGRADING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES

To upgrade your cluster with multi-architecture compute machines, use the **candidate-4.12** update channel. For more information, see "Understanding upgrade channels".

> **NOTE**
>
> Only OpenShift Container Platform clusters that are already using a multi-architecture payload can update with the **candidate-4.12** channel.

If you want to upgrade an existing cluster to support multi-architecture compute machines, you can perform an explicit upgrade command, as shown in the following procedure. This updates your current single-architecture cluster to a cluster that uses the multi-architecture payload.

Prerequisites

- You installed the OpenShift CLI (**oc**).

Procedure

- To manually upgrade your cluster, use the following command:

  ```
  $ oc adm upgrade --allow-explicit-upgrade --to-image <image-pullspec>  1
  ```

  1   You can access the **image-pullspec** object from the [mixed-arch mirrors page](#) in the **release.txt** file.

Additional resources

- [Understanding upgrade channels](#)

4.4. IMPORTING MANIFEST LISTS IN IMAGE STREAMS ON YOUR MULTI-ARCHITECTURE COMPUTE MACHINES

On an OpenShift Container Platform 4.12 cluster with multi-architecture compute machines, the image streams in the cluster do not import manifest lists automatically. You must manually change the default **importMode** option to the **PreserveOriginal** option in order to import the manifest list.

IMPORTANT

The **referencePolicy.type** field of your **ImageStream** object must be set to the **Source** type for this procedure to run successfully.

```
referencePolicy:
    type: Source
```

Prerequisites

- You installed the OpenShift Container Platform CLI (**oc**).

Procedure

- The following example command shows how to patch the **ImageStream** cli-artifacts so that the **cli-artifacts:latest** image stream tag is imported as a manifest list.

```
oc patch is/cli-artifacts -n openshift -p '{"spec":{"tags":[{"name":"latest","importPolicy":
{"importMode":"PreserveOriginal"}}]}}'
```

Verification

- You can check that the manifest lists imported properly by inspecting the image stream tag. The following command will list the individual architecture manifests for a particular tag.

```
oc get istag cli-artifacts:latest -n openshift -oyaml
```

If the **dockerImageManifests** object is present, then the manifest list import was successful.

Example output of the **dockerImageManifests** object

```
dockerImageManifests:
  - architecture: amd64
    digest:
sha256:16d4c96c52923a9968fbfa69425ec703aff711f1db822e4e9788bf5d2bee5d77
    manifestSize: 1252
    mediaType: application/vnd.docker.distribution.manifest.v2+json
    os: linux
  - architecture: arm64
    digest:
sha256:6ec8ad0d897bcdf727531f7d0b716931728999492709d19d8b09f0d90d57f626
    manifestSize: 1252
    mediaType: application/vnd.docker.distribution.manifest.v2+json
    os: linux
  - architecture: ppc64le
    digest:
sha256:65949e3a80349cdc42acd8c5b34cde6ebc3241eae8daaeea458498fedb359a6a
    manifestSize: 1252
    mediaType: application/vnd.docker.distribution.manifest.v2+json
    os: linux
  - architecture: s390x
    digest:
sha256:75f4fa21224b5d5d511bea8f92dfa8e1c00231e5c81ab95e83c3013d245d1719
```

```
manifestSize: 1252
mediaType: application/vnd.docker.distribution.manifest.v2+json
os: linux
```

CHAPTER 5. POSTINSTALLATION MACHINE CONFIGURATION TASKS

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.

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

5.1. UNDERSTANDING THE MACHINE CONFIG OPERATOR

5.1.1. Machine Config Operator

Purpose

The Machine Config Operator manages and applies configuration and updates of the base operating system and container runtime, including everything between the kernel and kubelet.

There are four components:

- **machine-config-server**: Provides Ignition configuration to new machines joining the cluster.

- **machine-config-controller**: Coordinates the upgrade of machines to the desired configurations defined by a **MachineConfig** object. Options are provided to control the upgrade for sets of machines individually.

- **machine-config-daemon**: Applies new machine configuration during update. Validates and verifies the state of the machine to the requested machine configuration.

- **machine-config**: Provides a complete source of machine configuration at installation, first start up, and updates for a machine.

> **IMPORTANT**
>
> Currently, there is no supported way to block or restrict the machine config server endpoint. The machine config server must be exposed to the network so that newly-provisioned machines, which have no existing configuration or state, are able to fetch their configuration. In this model, the root of trust is the certificate signing requests (CSR) endpoint, which is where the kubelet sends its certificate signing request for approval to join the cluster. Because of this, machine configs should not be used to distribute sensitive information, such as secrets and certificates.
>
> To ensure that the machine config server endpoints, ports 22623 and 22624, are secured in bare metal scenarios, customers must configure proper network policies.

Additional resources

- About the OpenShift SDN network plugin .

Project

openshift-machine-config-operator

5.1.2. 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.

5.1.2.1. 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.

- **fips**: Enable FIPS mode. FIPS should be set at installation-time setting and not a postinstallation procedure.
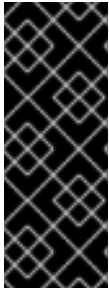
> **IMPORTANT**
>
> To enable FIPS mode for your cluster, you must run the installation program from a Red Hat Enterprise Linux (RHEL) computer configured to operate in FIPS mode. For more information about configuring FIPS mode on RHEL, see Installing the system in FIPS mode. The use of FIPS validated or Modules In Process cryptographic libraries is only supported on OpenShift Container Platform deployments on 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 postinstallation 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.

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*.

5.1.2.2. Project

See the openshift-machine-config-operator GitHub site for details.

5.1.3. 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."

You can mitigate the disruption caused by drain and reboot cycles by disabling control plane reboots. For more information, see "Disabling the Machine Config Operator from automatically rebooting."

Additional resources

- About the Machine Config Operator

- Disabling the Machine Config Operator from automatically rebooting

5.1.4. 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\"" ❶
    Reason:                1 nodes are reporting degraded status on sync
    Status:                True
    Type:                  NodeDegraded ❷
  ...
```

❶ 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.

❷ 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.

5.1.5. 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.

5.2. USING MACHINECONFIG 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.

5.2.1. 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: 4.12.0
metadata:
  name: 99-worker-chrony 1
  labels:
    machineconfiguration.openshift.io/role: worker 2
storage:
  files:
  - path: /etc/chrony.conf
    mode: 0644 3
    overwrite: true
    contents:
      inline: |
        pool 0.rhel.pool.ntp.org iburst 4
        driftfile /var/lib/chrony/drift
        makestep 1.0 3
        rtcsync
        logdir /var/log/chrony
```

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

3 Specify an octal value mode for the **mode** field in the machine config file. After creating the file and applying the changes, the **mode** is converted to a decimal value. You can check the YAML file with the command **oc get mc <mc-name> -o yaml**.

4 Specify any valid, reachable time source, such as the one provided by your DHCP server. Alternately, you can specify any of the following NTP servers: **1.rhel.pool.ntp.org**, **2.rhel.pool.ntp.org**, or **3.rhel.pool.ntp.org**.

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

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

3. Apply the configurations in one of two ways:

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

   - If the cluster is already running, apply the file:

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

Additional resources

- [Creating machine configs with Butane](#)

5.2.2. 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
   ```

5.2.3. 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**
  >
  > OpenShift Container Platform cgroups version 2 support 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 .

- **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 1
  name: 05-worker-kernelarg-selinuxpermissive 2
spec:
  kernelArguments:
    - enforcing=0 3
```

1   Applies the new kernel argument only to worker nodes.

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

3 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   v1.25.0
ip-10-0-136-243.ec2.internal   Ready                  master   34m   v1.25.0
ip-10-0-141-105.ec2.internal   Ready,SchedulingDisabled   worker   28m   v1.25.0
```

```
ip-10-0-142-249.ec2.internal   Ready          master   34m   v1.25.0
ip-10-0-153-11.ec2.internal    Ready          worker   28m   v1.25.0
ip-10-0-153-150.ec2.internal   Ready          master   34m   v1.25.0
```

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.

5.2.4. Enabling multipathing with kernel arguments on RHCOS

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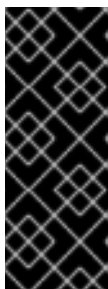
> **IMPORTANT**
>
> Enabling multipathing during installation is supported and recommended for nodes provisioned in OpenShift Container Platform 4.8 or higher. In setups where any I/O to non-optimized paths results in I/O system errors, you must enable multipathing at installation time. For more information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

> **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 4.12 cluster is installed or configured as a post-installation activity on a single VIOS host with "vSCSI" storage on {ibm-power-name} 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   v1.25.0
ip-10-0-136-243.ec2.internal   Ready                master  34m   v1.25.0
ip-10-0-141-105.ec2.internal   Ready,SchedulingDisabled   worker  28m   v1.25.0
ip-10-0-142-249.ec2.internal   Ready                master  34m   v1.25.0
ip-10-0-153-11.ec2.internal    Ready                worker  28m   v1.25.0
ip-10-0-153-150.ec2.internal   Ready                master  34m   v1.25.0
```

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.

5.2.5. 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, 4.12 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  v1.25.0
ip-10-0-146-92.us-east-2.compute.internal   Ready   worker   101m  v1.25.0
ip-10-0-169-2.us-east-2.compute.internal    Ready   worker   102m  v1.25.0
```

```
$ 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
```

5.2.6. 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: 4.12.0
   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

5.2.7. 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  v1.25.0
```

```
$ 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
```

5.2.8. 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**.
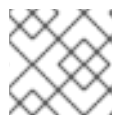
   > **NOTE**
   >
   > See "Creating machine configs with Butane" for information about Butane.

   Butane config file for custom firmware blob

   ```
   variant: openshift
   version: 4.12.0
   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](#)

5.3. 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.

5.3.1. 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:

   ```
   $ 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 1

...
```

1   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
    ```

5.3.2. 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**.
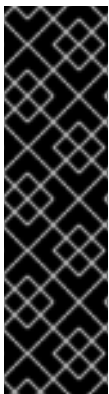
> **IMPORTANT**
>
> Support for the crun container runtime is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features 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 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: ''  ❶
 containerRuntimeConfig:
  logLevel: debug  ❷
  overlaySize: 8G  ❸
  defaultRuntime: "crun"  ❹
```

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

❷ 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**.

Prerequisite

- To enable crun, you must enable the **TechPreviewNoUpgrade** feature set.
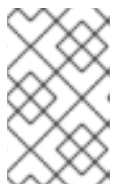
> **NOTE**
>
> Enabling the **TechPreviewNoUpgrade** feature set cannot be undone and prevents minor version updates. These feature sets are not recommended on production clusters.

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"
   ```

5.3.3. 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 6. POSTINSTALLATION CLUSTER TASKS

After installing OpenShift Container Platform, you can further expand and customize your cluster to your requirements.

6.1. AVAILABLE CLUSTER CUSTOMIZATIONS

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

> **NOTE**
>
> If you install your cluster on IBM Z, not all features and functions are available.

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

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

6.1.1. Cluster configuration resources

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

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

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

6.1.2. Operator configuration resources

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

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

6.1.3. Additional configuration resources

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

Resource name	Instance name	Namespace	Description
alertmanager.monitoring.coreos.com	**main**	**openshift-monitoring**	Controls the Alertmanager deployment parameters.

Resource name	Instance name	Namespace	Description
ingresscontroller.operator.openshift.io	**default**	**openshift-ingress-operator**	Configures Ingress Operator behavior such as domain, number of replicas, certificates, and controller placement.

6.1.4. Informational Resources

You use these resources to retrieve information about the cluster. Some configurations might require you to edit these resources directly.

Resource name	Instance name	Description
clusterversion.config.openshift.io	**version**	In OpenShift Container Platform 4.12, you must not customize the **ClusterVersion** resource for production clusters. Instead, follow the process to update a cluster.
dns.config.openshift.io	**cluster**	You cannot modify the DNS settings for your cluster. You can view the DNS Operator status.
infrastructure.config.openshift.io	**cluster**	Configuration details allowing the cluster to interact with its cloud provider.
network.config.openshift.io	**cluster**	You cannot modify your cluster networking after installation. To customize your network, follow the process to customize networking during installation.

6.2. UPDATING THE GLOBAL CLUSTER PULL SECRET

You can update the global pull secret for your cluster by either replacing the current pull secret or appending a new pull secret.

The procedure is required when users use a separate registry to store images than the registry used during installation.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Optional: To append a new pull secret to the existing pull secret, complete the following steps:

   a. Enter the following command to download the pull secret:

```
$ oc get secret/pull-secret -n openshift-config --template='{{index .data
".dockerconfigjson" | base64decode}}' ><pull_secret_location> 1
```

1   Provide the path to the pull secret file.

b. Enter the following command to add the new pull secret:

```
$ oc registry login --registry="<registry>" \ 1
--auth-basic="<username>:<password>" \ 2
--to=<pull_secret_location> 3
```

1   Provide the new registry. You can include multiple repositories within the same registry, for example: **--registry="<registry/my-namespace/my-repository>"**.

2   Provide the credentials of the new registry.

3   Provide the path to the pull secret file.

Alternatively, you can perform a manual update to the pull secret file.

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

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

1   Provide the path to the new pull secret file.

This update is rolled out to all nodes, which can take some time depending on the size of your cluster.

> **NOTE**
>
> As of OpenShift Container Platform 4.7.4, changes to the global pull secret no longer trigger a node drain or reboot.

6.3. ADDING WORKER NODES

After you deploy your OpenShift Container Platform cluster, you can add worker nodes to scale cluster resources. There are different ways you can add worker nodes depending on the installation method and the environment of your cluster.

6.3.1. Adding worker nodes to installer-provisioned infrastructure clusters

For installer-provisioned infrastructure clusters, you can manually or automatically scale the **MachineSet** object to match the number of available bare-metal hosts.

To add a bare-metal host, you must configure all network prerequisites, configure an associated **baremetalhost** object, then provision the worker node to the cluster. You can add a bare-metal host manually or by using the web console.

- Adding worker nodes using the web console

- [Adding worker nodes using YAML in the web console](#)

- [Manually adding a worker node to an installer-provisioned infrastructure cluster](#)

6.3.2. Adding worker nodes to user-provisioned infrastructure clusters

For user-provisioned infrastructure clusters, you can add worker nodes by using a RHEL or RHCOS ISO image and connecting it to your cluster using cluster Ignition config files. For RHEL worker nodes, the following example uses Ansible playbooks to add worker nodes to the cluster. For RHCOS worker nodes, the following example uses an ISO image and network booting to add worker nodes to the cluster.

- [Adding RHCOS worker nodes to a user-provisioned infrastructure cluster](#)

- [Adding RHEL worker nodes to a user-provisioned infrastructure cluster](#)

6.3.3. Adding worker nodes to clusters managed by the Assisted Installer

For clusters managed by the Assisted Installer, you can add worker nodes by using the Red Hat OpenShift Cluster Manager console, the Assisted Installer REST API or you can manually add worker nodes using an ISO image and cluster Ignition config files.

- [Adding worker nodes using the OpenShift Cluster Manager](#)

- [Adding worker nodes using the Assisted Installer REST API](#)

- [Manually adding worker nodes to a SNO cluster](#)

6.3.4. Adding worker nodes to clusters managed by the multicluster engine for Kubernetes

For clusters managed by the multicluster engine for Kubernetes, you can add worker nodes by using the dedicated multicluster engine console.

- [Scaling hosts to an infrastructure environment](#)

6.4. ADJUST WORKER NODES

If you incorrectly sized the worker nodes during deployment, adjust them by creating one or more new compute machine sets, scale them up, then scale the original compute machine set down before removing them.

6.4.1. Understanding the difference between compute machine sets and the machine config pool

MachineSet objects describe OpenShift Container Platform nodes with respect to the cloud or machine provider.

The **MachineConfigPool** object allows **MachineConfigController** components to define and provide the status of machines in the context of upgrades.

The **MachineConfigPool** object allows users to configure how upgrades are rolled out to the OpenShift Container Platform nodes in the machine config pool.

The **NodeSelector** object can be replaced with a reference to the **MachineSet** object.

6.4.2. Scaling a compute machine set manually

To add or remove an instance of a machine in a compute machine set, you can manually scale the compute machine set.

This guidance is relevant to fully automated, installer-provisioned infrastructure installations. Customized, user-provisioned infrastructure installations do not have compute machine sets.

Prerequisites

- Install an OpenShift Container Platform cluster and the **oc** command line.

- Log in to **oc** as a user with **cluster-admin** permission.

Procedure

1. View the compute machine sets that are in the cluster by running the following command:

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

   The compute machine sets are listed in the form of **<clusterid>-worker-<aws-region-az>**.

2. View the compute machines that are in the cluster by running the following command:

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

3. Set the annotation on the compute machine that you want to delete by running the following command:

   ```
   $ oc annotate machine/<machine_name> -n openshift-machine-api
   machine.openshift.io/delete-machine="true"
   ```

4. Scale the compute machine set by running one of the following commands:

   ```
   $ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
   ```

   Or:

   ```
   $ oc edit machineset <machineset> -n openshift-machine-api
   ```

   TIP

   You can alternatively apply the following YAML to scale the compute machine set:

   ```
   apiVersion: machine.openshift.io/v1beta1
   kind: MachineSet
   metadata:
     name: <machineset>
     namespace: openshift-machine-api
   spec:
     replicas: 2
   ```

You can scale the compute machine set up or down. It takes several minutes for the new machines to be available.

> **IMPORTANT**
>
> By default, the machine controller tries to drain the node that is backed by the machine until it succeeds. In some situations, such as with a misconfigured pod disruption budget, the drain operation might not be able to succeed. If the drain operation fails, the machine controller cannot proceed removing the machine.
>
> You can skip draining the node by annotating **machine.openshift.io/exclude-node-draining** in a specific machine.

Verification

- Verify the deletion of the intended machine by running the following command:

  ```
  $ oc get machines
  ```

6.4.3. The compute machine set deletion policy

Random, **Newest**, and **Oldest** are the three supported deletion options. The default is **Random**, meaning that random machines are chosen and deleted when scaling compute machine sets down. The deletion policy can be set according to the use case by modifying the particular compute machine set:

```
spec:
  deletePolicy: <delete_policy>
  replicas: <desired_replica_count>
```

Specific machines can also be prioritized for deletion by adding the annotation **machine.openshift.io/delete-machine=true** to the machine of interest, regardless of the deletion policy.

> **IMPORTANT**
>
> By default, the OpenShift Container Platform router pods are deployed on workers. Because the router is required to access some cluster resources, including the web console, do not scale the worker compute machine set to **0** unless you first relocate the router pods.

> **NOTE**
>
> Custom compute machine sets can be used for use cases requiring that services run on specific nodes and that those services are ignored by the controller when the worker compute machine sets are scaling down. This prevents service disruption.

6.4.4. Creating default cluster-wide node selectors

You can use default cluster-wide node selectors on pods together with labels on nodes to constrain all pods created in a cluster to specific nodes.

With cluster-wide node selectors, when you create a pod in that cluster, OpenShift Container Platform adds the default node selectors to the pod and schedules the pod on nodes with matching labels.

You configure cluster-wide node selectors by editing the Scheduler Operator custom resource (CR). You add labels to a node, a compute machine set, or a machine config. Adding the label to the compute machine set ensures that if the node or machine goes down, new nodes have the label. Labels added to a node or machine config do not persist if the node or machine goes down.

> **NOTE**
>
> You can add additional key/value pairs to a pod. But you cannot add a different value for a default key.

Procedure

To add a default cluster-wide node selector:

1. Edit the Scheduler Operator CR to add the default cluster-wide node selectors:

   ```
   $ oc edit scheduler cluster
   ```

   Example Scheduler Operator CR with a node selector

   ```
   apiVersion: config.openshift.io/v1
   kind: Scheduler
   metadata:
     name: cluster
   ...
   spec:
     defaultNodeSelector: type=user-node,region=east ❶
     mastersSchedulable: false
   ```

   ❶ Add a node selector with the appropriate **<key>:<value>** pairs.

   After making this change, wait for the pods in the **openshift-kube-apiserver** project to redeploy. This can take several minutes. The default cluster-wide node selector does not take effect until the pods redeploy.

2. Add labels to a node by using a compute machine set or editing the node directly:

   - Use a compute machine set to add labels to nodes managed by the compute machine set when a node is created:

     a. Run the following command to add labels to a **MachineSet** object:

        ```
        $ oc patch MachineSet <name> --type='json' -
        p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"<key>"="
        <value>","<key>"="<value>"}}]'  -n openshift-machine-api ❶
        ```

        ❶ Add a **<key>/<value>** pair for each label.

        For example:

        ```
        $ oc patch MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c --type='json' -
        p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"type":"user-
        node","region":"east"}}]'  -n openshift-machine-api
        ```

TIP

You can alternatively apply the following YAML to add labels to a compute machine set:

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  template:
    spec:
      metadata:
        labels:
          region: "east"
          type: "user-node"
```

b. Verify that the labels are added to the **MachineSet** object by using the **oc edit** command:
   For example:

```
$ oc edit MachineSet abc612-msrtw-worker-us-east-1c -n openshift-machine-api
```

Example **MachineSet** object

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
  ...
spec:
  ...
  template:
    metadata:
  ...
    spec:
      metadata:
        labels:
          region: east
          type: user-node
  ...
```

c. Redeploy the nodes associated with that compute machine set by scaling down to **0** and scaling up the nodes:
   For example:

```
$ oc scale --replicas=0 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

```
$ oc scale --replicas=1 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

d. When the nodes are ready and available, verify that the label is added to the nodes by using the **oc get** command:

```
$ oc get nodes -l <key>=<value>
```

For example:

```
$ oc get nodes -l type=user-node
```

Example output

```
NAME                                STATUS   ROLES    AGE   VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-c-vmqzp   Ready    worker   61s   v1.25.0
```

- Add labels directly to a node:

  a. Edit the **Node** object for the node:

  ```
  $ oc label nodes <name> <key>=<value>
  ```

  For example, to label a node:

  ```
  $ oc label nodes ci-ln-l8nry52-f76d1-hl7m7-worker-b-tgq49 type=user-node
  region=east
  ```

  TIP

  You can alternatively apply the following YAML to add labels to a node:

  ```
  kind: Node
  apiVersion: v1
  metadata:
    name: <node_name>
    labels:
      type: "user-node"
      region: "east"
  ```

  b. Verify that the labels are added to the node using the **oc get** command:

  ```
  $ oc get nodes -l <key>=<value>,<key>=<value>
  ```

  For example:

  ```
  $ oc get nodes -l type=user-node,region=east
  ```

  Example output

  ```
  NAME                                STATUS   ROLES    AGE   VERSION
  ci-ln-l8nry52-f76d1-hl7m7-worker-b-tgq49   Ready    worker   17m   v1.25.0
  ```

6.4.5. Creating user workloads in AWS Local Zones

After you create an Amazon Web Service (AWS) Local Zone environment, and you deploy your cluster, you can use edge worker nodes to create user workloads in Local Zone subnets.

After the **openshift-installer** creates the cluster, the installation program automatically specifies a taint effect of **NoSchedule** to each edge worker node. This means that a scheduler does not add a new pod, or deployment, to a node if the pod does not match the specified tolerations for a taint. You can modify the taint for better control over how each node creates a workload in each Local Zone subnet.

The **openshift-installer** creates the compute machine set manifests file with **node-role.kubernetes.io/edge** and **node-role.kubernetes.io/worker** labels applied to each edge worker node that is located in a Local Zone subnet.

Prerequisites

- You have access to the OpenShift CLI (**oc**).

- You deployed your cluster in a Virtual Private Cloud (VPC) with defined Local Zone subnets.

- You ensured that the compute machine set for the edge workers on Local Zone subnets specifies the taints for **node-role.kubernetes.io/edge**.

Procedure

1. Create a **deployment** resource YAML file for an example application to be deployed in the edge worker node that operates in a Local Zone subnet. Ensure that you specify the correct tolerations that match the taints for the edge worker node.

   Example of a configured deployment resource for an edge worker node that operates in a Local Zone subnet

   ```
   kind: Namespace
   apiVersion: v1
   metadata:
     name: <local_zone_application_namespace>
   ---
   kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: <pvc_name>
     namespace: <local_zone_application_namespace>
   spec:
     accessModes:
       - ReadWriteOnce
     resources:
       requests:
         storage: 10Gi
     storageClassName: gp2-csi ❶
     volumeMode: Filesystem
   ---
   apiVersion: apps/v1
   kind: Deployment ❷
   metadata:
     name: <local_zone_application> ❸
     namespace: <local_zone_application_namespace> ❹
   spec:
   ```

```
    selector:
      matchLabels:
        app: <local_zone_application>
    replicas: 1
    template:
      metadata:
        labels:
          app: <local_zone_application>
          zone-group: ${ZONE_GROUP_NAME} 5
      spec:
        securityContext:
          seccompProfile:
            type: RuntimeDefault
        nodeSelector: 6
          machine.openshift.io/zone-group: ${ZONE_GROUP_NAME}
        tolerations: 7
        - key: "node-role.kubernetes.io/edge"
          operator: "Equal"
          value: ""
          effect: "NoSchedule"
        containers:
          - image: openshift/origin-node
            command:
             - "/bin/socat"
            args:
              - TCP4-LISTEN:8080,reuseaddr,fork
              - EXEC:/bin/bash -c \"printf \\\"HTTP/1.0 200 OK\r\n\r\n\\\"; sed -e \\\"/^\r/q\\\"\"
            imagePullPolicy: Always
            name: echoserver
            ports:
              - containerPort: 8080
            volumeMounts:
              - mountPath: "/mnt/storage"
                name: data
        volumes:
        - name: data
          persistentVolumeClaim:
            claimName: <pvc_name>
```

1. **storageClassName**: For the Local Zone configuration, you must specify **gp2-csi**.

2. **kind**: Defines the **deployment** resource.

3. **name**: Specifies the name of your Local Zone application. For example, **local-zone-demo-app-nyc-1**.

4. **namespace:** Defines the namespace for the AWS Local Zone where you want to run the user workload. For example: **local-zone-app-nyc-1a**.

5. **zone-group**: Defines the group to where a zone belongs. For example, **us-east-1-iah-1**.

6. **nodeSelector**: Targets edge worker nodes that match the specified labels.

7. **tolerations**: Sets the values that match with the **taints** defined on the **MachineSet** manifest for the Local Zone node.

2. Create a **service** resource YAML file for the node. This resource exposes a pod from a targeted edge worker node to services that run inside your Local Zone network.

Example of a configured service resource for an edge worker node that operates in a Local Zone subnet

```
apiVersion: v1
kind: Service ❶
metadata:
  name:  <local_zone_application>
  namespace: <local_zone_application_namespace>
spec:
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: NodePort
  selector: ❷
    app: <local_zone_application>
```

❶ **kind**: Defines the **service** resource.

❷ **selector:** Specifies the label type applied to managed pods.

Next steps

- Optional: Use the AWS Load Balancer (ALB) Operator to expose a pod from a targeted edge worker node to services that run inside a Local Zone subnet from a public network. See Installing the AWS Load Balancer Operator .

Additional resources

- Installing a cluster using AWS Local Zones

- Understanding taints and tolerations

- Using taints and tolerations to control logging pod placement

6.5. IMPROVING CLUSTER STABILITY IN HIGH LATENCY ENVIRONMENTS USING WORKER LATENCY PROFILES

If the cluster administrator has performed latency tests for platform verification, they can discover the need to adjust the operation of the cluster to ensure stability in cases of high latency. The cluster administrator need change only one parameter, recorded in a file, which controls four parameters affecting how supervisory processes read status and interpret the health of the cluster. Changing only the one parameter provides cluster tuning in an easy, supportable manner.

The **Kubelet** process provides the starting point for monitoring cluster health. The **Kubelet** sets status values for all nodes in the OpenShift Container Platform cluster. The Kubernetes Controller Manager (**kube controller**) reads the status values every 10 seconds, by default. If the **kube controller** cannot read a node status value, it loses contact with that node after a configured period. The default behavior is:

1. The node controller on the control plane updates the node health to **Unhealthy** and marks the node **Ready** condition `Unknown`.

2. In response, the scheduler stops scheduling pods to that node.

3. The Node Lifecycle Controller adds a **node.kubernetes.io/unreachable** taint with a **NoExecute** effect to the node and schedules any pods on the node for eviction after five minutes, by default.

This behavior can cause problems if your network is prone to latency issues, especially if you have nodes at the network edge. In some cases, the Kubernetes Controller Manager might not receive an update from a healthy node due to network latency. The **Kubelet** evicts pods from the node even though the node is healthy.

To avoid this problem, you can use *worker latency profiles* to adjust the frequency that the **Kubelet** and the Kubernetes Controller Manager wait for status updates before taking action. These adjustments help to ensure that your cluster runs properly if network latency between the control plane and the worker nodes is not optimal.

These worker latency profiles contain three sets of parameters that are pre-defined with carefully tuned values to control the reaction of the cluster to increased latency. No need to experimentally find the best values manually.

You can configure worker latency profiles when installing a cluster or at any time you notice increased latency in your cluster network.

6.5.1. Understanding worker latency profiles

Worker latency profiles are four different categories of carefully-tuned parameters. The four parameters which implement these values are **node-status-update-frequency**, **node-monitor-grace-period**, **default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds**. These parameters can use values which allow you control the reaction of the cluster to latency issues without needing to determine the best values using manual methods.
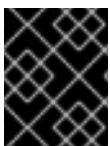
> **IMPORTANT**
>
> Setting these parameters manually is not supported. Incorrect parameter settings adversely affect cluster stability.

All worker latency profiles configure the following parameters:

node-status-update-frequency

Specifies how often the kubelet posts node status to the API server.

node-monitor-grace-period

Specifies the amount of time in seconds that the Kubernetes Controller Manager waits for an update from a kubelet before marking the node unhealthy and adding the **node.kubernetes.io/not-ready** or **node.kubernetes.io/unreachable** taint to the node.

default-not-ready-toleration-seconds

Specifies the amount of time in seconds after marking a node unhealthy that the Kube API Server Operator waits before evicting pods from that node.

default-unreachable-toleration-seconds

Specifies the amount of time in seconds after marking a node unreachable that the Kube API Server Operator waits before evicting pods from that node.

The following Operators monitor the changes to the worker latency profiles and respond accordingly:

- The Machine Config Operator (MCO) updates the **node-status-update-frequency** parameter on the worker nodes.

- The Kubernetes Controller Manager updates the **node-monitor-grace-period** parameter on the control plane nodes.

- The Kubernetes API Server Operator updates the **default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds** parameters on the control plane nodes.

While the default configuration works in most cases, OpenShift Container Platform offers two other worker latency profiles for situations where the network is experiencing higher latency than usual. The three worker latency profiles are described in the following sections:

Default worker latency profile

With the **Default** profile, each **Kubelet** updates it's status every 10 seconds (**node-status-update-frequency**). The **Kube Controller Manager** checks the statuses of **Kubelet** every 5 seconds (**node-monitor-grace-period**).
The Kubernetes Controller Manager waits 40 seconds for a status update from **Kubelet** before considering the **Kubelet** unhealthy. If no status is made available to the Kubernetes Controller Manager, it then marks the node with the **node.kubernetes.io/not-ready** or **node.kubernetes.io/unreachable** taint and evicts the pods on that node.

If a pod on that node has the **NoExecute** taint, the pod is run according to **tolerationSeconds**. If the pod has no taint, it will be evicted in 300 seconds (**default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds** settings of the **Kube API Server**).

Profile	Component	Parameter	Value
Default	kubelet	**node-status-update-frequency**	10s
	Kubelet Controller Manager	**node-monitor-grace-period**	40s
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	300s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	300s

Medium worker latency profile

Use the **MediumUpdateAverageReaction** profile if the network latency is slightly higher than usual. The **MediumUpdateAverageReaction** profile reduces the frequency of kubelet updates to 20 seconds and changes the period that the Kubernetes Controller Manager waits for those updates to 2 minutes. The pod eviction period for a pod on that node is reduced to 60 seconds. If the pod has the **tolerationSeconds** parameter, the eviction waits for the period specified by that parameter.

The Kubernetes Controller Manager waits for 2 minutes to consider a node unhealthy. In another minute, the eviction process starts.

Profile	Component	Parameter	Value
MediumUpdateAverageReaction	kubelet	**node-status-update-frequency**	20s
	Kubelet Controller Manager	**node-monitor-grace-period**	2m
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	60s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	60s

Low worker latency profile

Use the **LowUpdateSlowReaction** profile if the network latency is extremely high.
The **LowUpdateSlowReaction** profile reduces the frequency of kubelet updates to 1 minute and changes the period that the Kubernetes Controller Manager waits for those updates to 5 minutes. The pod eviction period for a pod on that node is reduced to 60 seconds. If the pod has the **tolerationSeconds** parameter, the eviction waits for the period specified by that parameter.

The Kubernetes Controller Manager waits for 5 minutes to consider a node unhealthy. In another minute, the eviction process starts.

Profile	Component	Parameter	Value
LowUpdateSlowReaction	kubelet	**node-status-update-frequency**	1m
	Kubelet Controller Manager	**node-monitor-grace-period**	5m
	Kubernetes API Server Operator	**default-not-ready-toleration-seconds**	60s
	Kubernetes API Server Operator	**default-unreachable-toleration-seconds**	60s

6.5.2. Using and changing worker latency profiles

To change a worker latency profile to deal with network latency, edit the **node.config** object to add the name of the profile. You can change the profile at any time as latency increases or decreases.

You must move one worker latency profile at a time. For example, you cannot move directly from the **Default** profile to the **LowUpdateSlowReaction** worker latency profile. You must move from the **Default** worker latency profile to the **MediumUpdateAverageReaction** profile first, then to **LowUpdateSlowReaction**. Similarly, when returning to the **Default** profile, you must move from the low profile to the medium profile first, then to **Default**.

> **NOTE**
>
> You can also configure worker latency profiles upon installing an OpenShift Container Platform cluster.

Procedure

To move from the default worker latency profile:

1. Move to the medium worker latency profile:

   a. Edit the **node.config** object:

   ```
   $ oc edit nodes.config/cluster
   ```

   b. Add **spec.workerLatencyProfile: MediumUpdateAverageReaction**:

   Example node.config object

   ```
   apiVersion: config.openshift.io/v1
   kind: Node
   metadata:
     annotations:
       include.release.openshift.io/ibm-cloud-managed: "true"
       include.release.openshift.io/self-managed-high-availability: "true"
       include.release.openshift.io/single-node-developer: "true"
       release.openshift.io/create-only: "true"
     creationTimestamp: "2022-07-08T16:02:51Z"
     generation: 1
     name: cluster
     ownerReferences:
     - apiVersion: config.openshift.io/v1
       kind: ClusterVersion
       name: version
       uid: 36282574-bf9f-409e-a6cd-3032939293eb
     resourceVersion: "1865"
     uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
   spec:
     workerLatencyProfile: MediumUpdateAverageReaction    ❶

   # ...
   ```

   ❶ Specifies the medium worker latency policy.

   Scheduling on each worker node is disabled as the change is being applied.

2. Optional: Move to the low worker latency profile:

   a. Edit the **node.config** object:

      ```
      $ oc edit nodes.config/cluster
      ```

   b. Change the **spec.workerLatencyProfile** value to **LowUpdateSlowReaction**:

   Example node.config object

   ```
   apiVersion: config.openshift.io/v1
   kind: Node
   metadata:
     annotations:
       include.release.openshift.io/ibm-cloud-managed: "true"
       include.release.openshift.io/self-managed-high-availability: "true"
       include.release.openshift.io/single-node-developer: "true"
       release.openshift.io/create-only: "true"
     creationTimestamp: "2022-07-08T16:02:51Z"
     generation: 1
     name: cluster
     ownerReferences:
     - apiVersion: config.openshift.io/v1
       kind: ClusterVersion
       name: version
       uid: 36282574-bf9f-409e-a6cd-3032939293eb
     resourceVersion: "1865"
     uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
   spec:
     workerLatencyProfile: LowUpdateSlowReaction ❶

   # ...
   ```

   ❶ Specifies use of the low worker latency policy.

Scheduling on each worker node is disabled as the change is being applied.

Verification

- When all nodes return to the **Ready** condition, you can use the following command to look in the Kubernetes Controller Manager to ensure it was applied:

  ```
  $ oc get KubeControllerManager -o yaml | grep -i workerlatency -A 5 -B 5
  ```

Example output

```
# ...
    - lastTransitionTime: "2022-07-11T19:47:10Z"
      reason: ProfileUpdated
      status: "False"
      type: WorkerLatencyProfileProgressing
    - lastTransitionTime: "2022-07-11T19:47:10Z" ❶
      message: all static pod revision(s) have updated latency profile
```

```
      reason: ProfileUpdated
      status: "True"
      type: WorkerLatencyProfileComplete
    - lastTransitionTime: "2022-07-11T19:20:11Z"
      reason: AsExpected
      status: "False"
      type: WorkerLatencyProfileDegraded
    - lastTransitionTime: "2022-07-11T19:20:36Z"
      status: "False"
# ...
```

1 Specifies that the profile is applied and active.

To change the medium profile to default or change the default to medium, edit the **node.config** object and set the **spec.workerLatencyProfile** parameter to the appropriate value.

6.6. MANAGING CONTROL PLANE MACHINES

Control plane machine sets provide management capabilities for control plane machines that are similar to what compute machine sets provide for compute machines. The availability and initial status of control plane machine sets on your cluster depend on your cloud provider and the version of OpenShift Container Platform that you installed. For more information, see Getting started with control plane machine sets.

6.7. CREATING INFRASTRUCTURE MACHINE SETS FOR PRODUCTION ENVIRONMENTS

You can create a compute machine set to create machines that host only infrastructure components, such as the default router, the integrated container image registry, and components for cluster metrics and monitoring. These infrastructure machines are not counted toward the total number of subscriptions that are required to run the environment.

In a production deployment, it is recommended that you deploy at least three compute machine sets to hold infrastructure components. Both OpenShift Logging and Red Hat OpenShift Service Mesh deploy Elasticsearch, which requires three instances to be installed on different nodes. Each of these nodes can be deployed to different availability zones for high availability. A configuration like this requires three different compute machine sets, one for each availability zone. In global Azure regions that do not have multiple availability zones, you can use availability sets to ensure high availability.

For information on infrastructure nodes and which components can run on infrastructure nodes, see Creating infrastructure machine sets.

To create an infrastructure node, you can use a machine set, assign a label to the nodes , or use a machine config pool.

For sample machine sets that you can use with these procedures, see Creating machine sets for different clouds.

Applying a specific node selector to all infrastructure components causes OpenShift Container Platform to schedule those workloads on nodes with that label .

6.7.1. Creating a compute machine set

In addition to the compute machine sets created by the installation program, you can create your own to dynamically manage the machine compute resources for specific workloads of your choice.

Prerequisites

- Deploy an OpenShift Container Platform cluster.

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

- Log in to **oc** as a user with **cluster-admin** permission.

Procedure

1. Create a new YAML file that contains the compute machine set custom resource (CR) sample and is named **<file_name>.yaml**.
   Ensure that you set the **<clusterID>** and **<role>** parameter values.

2. Optional: If you are not sure which value to set for a specific field, you can check an existing compute machine set from your cluster.

   a. To list the compute machine sets in your cluster, run the following command:

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

   Example output

   ```
   NAME                          DESIRED  CURRENT  READY  AVAILABLE  AGE
   agl030519-vplxk-worker-us-east-1a  1      1      1    1         55m
   agl030519-vplxk-worker-us-east-1b  1      1      1    1         55m
   agl030519-vplxk-worker-us-east-1c  1      1      1    1         55m
   agl030519-vplxk-worker-us-east-1d  0      0                     55m
   agl030519-vplxk-worker-us-east-1e  0      0                     55m
   agl030519-vplxk-worker-us-east-1f  0      0                     55m
   ```

   b. To view values of a specific compute machine set custom resource (CR), run the following command:

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

   Example output

   ```
   apiVersion: machine.openshift.io/v1beta1
   kind: MachineSet
   metadata:
     labels:
       machine.openshift.io/cluster-api-cluster: <infrastructure_id>  ❶
     name: <infrastructure_id>-<role>  ❷
     namespace: openshift-machine-api
   spec:
     replicas: 1
     selector:
       matchLabels:
         machine.openshift.io/cluster-api-cluster: <infrastructure_id>
   ```

```
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
      template:
       metadata:
        labels:
          machine.openshift.io/cluster-api-cluster: <infrastructure_id>
          machine.openshift.io/cluster-api-machine-role: <role>
          machine.openshift.io/cluster-api-machine-type: <role>
          machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
       spec:
        providerSpec: 3
          ...
```

1 The cluster infrastructure ID.

2 A default node label.

> **NOTE**
>
> For clusters that have user-provisioned infrastructure, a compute machine set can only create **worker** and **infra** type machines.

3 The values in the **<providerSpec>** section of the compute machine set CR are platform-specific. For more information about **<providerSpec>** parameters in the CR, see the sample compute machine set CR configuration for your provider.

3. Create a **MachineSet** CR by running the following command:

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

Verification

- View the list of compute machine sets by running the following command:

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

Example output

```
NAME                            DESIRED  CURRENT  READY  AVAILABLE  AGE
agl030519-vplxk-infra-us-east-1a   1        1        1      1          11m
agl030519-vplxk-worker-us-east-1a  1        1        1      1          55m
agl030519-vplxk-worker-us-east-1b  1        1        1      1          55m
agl030519-vplxk-worker-us-east-1c  1        1        1      1          55m
agl030519-vplxk-worker-us-east-1d  0        0                          55m
agl030519-vplxk-worker-us-east-1e  0        0                          55m
agl030519-vplxk-worker-us-east-1f  0        0                          55m
```

When the new compute machine set is available, the **DESIRED** and **CURRENT** values match. If the compute machine set is not available, wait a few minutes and run the command again.

6.7.2. Creating an infrastructure node

> **IMPORTANT**
>
> See Creating infrastructure machine sets for installer-provisioned infrastructure environments or for any cluster where the control plane nodes are managed by the machine API.

Requirements of the cluster dictate that infrastructure, also called **infra** nodes, be provisioned. The installer only provides provisions for control plane and worker nodes. Worker nodes can be designated as infrastructure nodes or application, also called **app**, nodes through labeling.

Procedure

1. Add a label to the worker node that you want to act as application node:

   ```
   $ oc label node <node-name> node-role.kubernetes.io/app=""
   ```

2. Add a label to the worker nodes that you want to act as infrastructure nodes:

   ```
   $ oc label node <node-name> node-role.kubernetes.io/infra=""
   ```

3. Check to see if applicable nodes now have the **infra** role and **app** roles:

   ```
   $ oc get nodes
   ```

4. Create a default cluster-wide node selector. The default node selector is applied to pods created in all namespaces. This creates an intersection with any existing node selectors on a pod, which additionally constrains the pod's selector.

   > **IMPORTANT**
   >
   > If the default node selector key conflicts with the key of a pod's label, then the default node selector is not applied.
   >
   > However, do not set a default node selector that might cause a pod to become unschedulable. For example, setting the default node selector to a specific node role, such as **node-role.kubernetes.io/infra=""**, when a pod's label is set to a different node role, such as **node-role.kubernetes.io/master=""**, can cause the pod to become unschedulable. For this reason, use caution when setting the default node selector to specific node roles.
   >
   > You can alternatively use a project node selector to avoid cluster-wide node selector key conflicts.

   a. Edit the **Scheduler** object:

      ```
      $ oc edit scheduler cluster
      ```

   b. Add the **defaultNodeSelector** field with the appropriate node selector:

      ```
      apiVersion: config.openshift.io/v1
      kind: Scheduler
      metadata:
        name: cluster
      ```

```
spec:
  defaultNodeSelector: node-role.kubernetes.io/infra="" 1
  # ...
```

1   This example node selector deploys pods on infrastructure nodes by default.

c. Save the file to apply the changes.

You can now move infrastructure resources to the newly labeled **infra** nodes.

Additional resources

- For information on how to configure project node selectors to avoid cluster-wide node selector key conflicts, see Project node selectors .

6.7.3. Creating a machine config pool for infrastructure machines

If you need infrastructure machines to have dedicated configurations, you must create an infra pool.

Procedure

1. Add a label to the node you want to assign as the infra node with a specific label:

```
$ oc label node <node_name> <label>
```

```
$ oc label node ci-ln-n8mqwr2-f76d1-xscn2-worker-c-6fmtx node-role.kubernetes.io/infra=
```

2. Create a machine config pool that contains both the worker role and your custom role as machine config selector:

```
$ cat infra.mcp.yaml
```

Example output

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: infra
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,infra]} 1
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/infra: "" 2
```

1   Add the worker role and your custom role.

2   Add the label you added to the node as a **nodeSelector**.

> **NOTE**
>
> Custom machine config pools inherit machine configs from the worker pool. Custom pools use any machine config targeted for the worker pool, but add the ability to also deploy changes that are targeted at only the custom pool. Because a custom pool inherits resources from the worker pool, any change to the worker pool also affects the custom pool.

3. After you have the YAML file, you can create the machine config pool:

   ```
   $ oc create -f infra.mcp.yaml
   ```

4. Check the machine configs to ensure that the infrastructure configuration rendered successfully:

   ```
   $ oc get machineconfig
   ```

Example output

```
NAME                                                   GENERATEDBYCONTROLLER
IGNITIONVERSION   CREATED
00-master                                               365c1cfd14de5b0e3b85e0fc815b0060f36ab955
3.2.0          31d
00-worker                                               365c1cfd14de5b0e3b85e0fc815b0060f36ab955
3.2.0          31d
01-master-container-runtime
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0          31d
01-master-kubelet                                       365c1cfd14de5b0e3b85e0fc815b0060f36ab955
3.2.0          31d
01-worker-container-runtime
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0          31d
01-worker-kubelet                                       365c1cfd14de5b0e3b85e0fc815b0060f36ab955
3.2.0          31d
99-master-1ae2a1e0-a115-11e9-8f14-005056899d54-registries
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0          31d
99-master-ssh                                                              3.2.0          31d
99-worker-1ae64748-a115-11e9-8f14-005056899d54-registries
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0          31d
99-worker-ssh                                                              3.2.0          31d
rendered-infra-4e48906dca84ee702959c71a53ee80e7
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0          23m
rendered-master-072d4b2da7f88162636902b074e9e28e
5b6fb8349a29735e48446d435962dec4547d3090   3.2.0          31d
rendered-master-3e88ec72aed3886dec061df60d16d1af
02c07496ba0417b3e12b78fb32baf6293d314f79   3.2.0          31d
rendered-master-419bee7de96134963a15fdf9dd473b25
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0          17d
rendered-master-53f5c91c7661708adce18739cc0f40fb
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0          13d
rendered-master-a6a357ec18e5bce7f5ac426fc7c5ffcd
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0          7d3h
rendered-master-dc7f874ec77fc4b969674204332da037
5b6fb8349a29735e48446d435962dec4547d3090   3.2.0          31d
rendered-worker-1a75960c52ad18ff5dfa6674eb7e533d
5b6fb8349a29735e48446d435962dec4547d3090   3.2.0          31d
```

```
rendered-worker-2640531be11ba43c61d72e82dc634ce6
5b6fb8349a29735e48446d435962dec4547d3090   3.2.0        31d
rendered-worker-4e48906dca84ee702959c71a53ee80e7
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0        7d3h
rendered-worker-4f110718fe88e5f349987854a1147755
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0        17d
rendered-worker-afc758e194d6188677eb837842d3b379
02c07496ba0417b3e12b78fb32baf6293d314f79   3.2.0        31d
rendered-worker-daa08cc1e8f5fcdeba24de60cd955cc3
365c1cfd14de5b0e3b85e0fc815b0060f36ab955   3.2.0        13d
```

You should see a new machine config, with the **rendered-infra-*** prefix.

5. Optional: To deploy changes to a custom pool, create a machine config that uses the custom pool name as the label, such as **infra**. Note that this is not required and only shown for instructional purposes. In this manner, you can apply any custom configurations specific to only your infra nodes.

> **NOTE**
>
> After you create the new machine config pool, the MCO generates a new rendered config for that pool, and associated nodes of that pool reboot to apply the new configuration.

   a. Create a machine config:

   ```
   $ cat infra.mc.yaml
   ```

   Example output

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     name: 51-infra
     labels:
       machineconfiguration.openshift.io/role: infra ❶
   spec:
     config:
       ignition:
         version: 3.2.0
       storage:
         files:
         - path: /etc/infratest
           mode: 0644
           contents:
             source: data:,infra
   ```

   ❶ Add the label you added to the node as a **nodeSelector**.

   b. Apply the machine config to the infra–labeled nodes:

   ```
   $ oc create -f infra.mc.yaml
   ```

6. Confirm that your new machine config pool is available:

```
$ oc get mcp
```

Example output

```
NAME     CONFIG                                        UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
infra    rendered-infra-60e35c2e99f42d976e084fa94da4d0fc    True      False      False      1
1              1                   0                   4m20s
master   rendered-master-9360fdb895d4c131c7c4bebbae099c90   True      False      False
3              3                   3                   0          91m
worker   rendered-worker-60e35c2e99f42d976e084fa94da4d0fc   True      False      False
2              2                   2                   0          91m
```

In this example, a worker node was changed to an infra node.

Additional resources

- See Node configuration management with machine config pools for more information on grouping infra machines in a custom pool.

6.8. ASSIGNING MACHINE SET RESOURCES TO INFRASTRUCTURE NODES

After creating an infrastructure machine set, the **worker** and **infra** roles are applied to new infra nodes. Nodes with the **infra** role are not counted toward the total number of subscriptions that are required to run the environment, even when the **worker** role is also applied.

However, when an infra node is assigned the worker role, there is a chance that user workloads can get assigned inadvertently to the infra node. To avoid this, you can apply a taint to the infra node and tolerations for the pods that you want to control.

6.8.1. Binding infrastructure node workloads using taints and tolerations

If you have an infra node that has the **infra** and **worker** roles assigned, you must configure the node so that user workloads are not assigned to it.

> **IMPORTANT**
>
> It is recommended that you preserve the dual **infra,worker** label that is created for infra nodes and use taints and tolerations to manage nodes that user workloads are scheduled on. If you remove the **worker** label from the node, you must create a custom pool to manage it. A node with a label other than **master** or **worker** is not recognized by the MCO without a custom pool. Maintaining the **worker** label allows the node to be managed by the default worker machine config pool, if no custom pools that select the custom label exists. The **infra** label communicates to the cluster that it does not count toward the total number of subscriptions.

Prerequisites

- Configure additional **MachineSet** objects in your OpenShift Container Platform cluster.

Procedure

1. Add a taint to the infra node to prevent scheduling user workloads on it:

   a. Determine if the node has the taint:

   ```
   $ oc describe nodes <node_name>
   ```

   Sample output

   ```
   oc describe node ci-ln-iyhx092-f76d1-nvdfm-worker-b-wln2l
   Name:           ci-ln-iyhx092-f76d1-nvdfm-worker-b-wln2l
   Roles:          worker
    ...
   Taints:         node-role.kubernetes.io/infra:NoSchedule
    ...
   ```

   This example shows that the node has a taint. You can proceed with adding a toleration to your pod in the next step.

   b. If you have not configured a taint to prevent scheduling user workloads on it:

   ```
   $ oc adm taint nodes <node_name> <key>=<value>:<effect>
   ```

   For example:

   ```
   $ oc adm taint nodes node1 node-role.kubernetes.io/infra=reserved:NoExecute
   ```

   TIP

   You can alternatively apply the following YAML to add the taint:

   ```
   kind: Node
   apiVersion: v1
   metadata:
     name: <node_name>
     labels:
       ...
   spec:
     taints:
       - key: node-role.kubernetes.io/infra
         effect: NoExecute
         value: reserved
    ...
   ```

   This example places a taint on **node1** that has key **node-role.kubernetes.io/infra** and taint effect **NoSchedule**. Nodes with the **NoSchedule** effect schedule only pods that tolerate the taint, but allow existing pods to remain scheduled on the node.

   > **NOTE**
   >
   > If a descheduler is used, pods violating node taints could be evicted from the cluster.

2. Add tolerations for the pod configurations you want to schedule on the infra node, like router, registry, and monitoring workloads. Add the following code to the **Pod** object specification:

```
tolerations:
  - effect: NoExecute 1
    key: node-role.kubernetes.io/infra 2
    operator: Exists 3
    value: reserved 4
```

1   Specify the effect that you added to the node.

2   Specify the key that you added to the node.

3   Specify the **Exists** Operator to require a taint with the key **node-role.kubernetes.io/infra** to be present on the node.

4   Specify the value of the key-value pair taint that you added to the node.

This toleration matches the taint created by the **oc adm taint** command. A pod with this toleration can be scheduled onto the infra node.

> **NOTE**
>
> Moving pods for an Operator installed via OLM to an infra node is not always possible. The capability to move Operator pods depends on the configuration of each Operator.

3. Schedule the pod to the infra node using a scheduler. See the documentation for *Controlling pod placement onto nodes* for details.

Additional resources

- See [Controlling pod placement using the scheduler](#) for general information on scheduling a pod to a node.

6.9. MOVING RESOURCES TO INFRASTRUCTURE MACHINE SETS

Some of the infrastructure resources are deployed in your cluster by default. You can move them to the infrastructure machine sets that you created.

6.9.1. Moving the router

You can deploy the router pod to a different compute machine set. By default, the pod is deployed to a worker node.

Prerequisites

- Configure additional compute machine sets in your OpenShift Container Platform cluster.

Procedure

1. View the **IngressController** custom resource for the router Operator:

```
$ oc get ingresscontroller default -n openshift-ingress-operator -o yaml
```

The command output resembles the following text:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: 2019-04-18T12:35:39Z
  finalizers:
  - ingresscontroller.operator.openshift.io/finalizer-ingresscontroller
  generation: 1
  name: default
  namespace: openshift-ingress-operator
  resourceVersion: "11341"
  selfLink: /apis/operator.openshift.io/v1/namespaces/openshift-ingress-
operator/ingresscontrollers/default
  uid: 79509e05-61d6-11e9-bc55-02ce4781844a
spec: {}
status:
  availableReplicas: 2
  conditions:
  - lastTransitionTime: 2019-04-18T12:36:15Z
    status: "True"
    type: Available
  domain: apps.<cluster>.example.com
  endpointPublishingStrategy:
    type: LoadBalancerService
  selector: ingresscontroller.operator.openshift.io/deployment-ingresscontroller=default
```

2. Edit the **ingresscontroller** resource and change the **nodeSelector** to use the **infra** label:

```
$ oc edit ingresscontroller default -n openshift-ingress-operator
```

```
spec:
  nodePlacement:
    nodeSelector:  ❶
      matchLabels:
        node-role.kubernetes.io/infra: ""
    tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved
```

❶ Add a **nodeSelector** parameter with the appropriate value to the component you want to move. You can use a **nodeSelector** in the format shown or use **<key>: <value>** pairs, based on the value specified for the node. If you added a taint to the infrastructure node, also add a matching toleration.

3. Confirm that the router pod is running on the **infra** node.

   a. View the list of router pods and note the node name of the running pod:

```
$ oc get pod -n openshift-ingress -o wide
```

Example output

```
NAME                          READY   STATUS        RESTARTS   AGE     IP           NODE
NOMINATED NODE   READINESS GATES
router-default-86798b4b5d-bdlvd   1/1     Running       0          28s     10.130.2.4   ip-10-
0-217-226.ec2.internal   <none>          <none>
router-default-955d875f4-255g8    0/1     Terminating   0          19h     10.129.2.4   ip-10-
0-148-172.ec2.internal   <none>          <none>
```

In this example, the running pod is on the **ip-10-0-217-226.ec2.internal** node.

b. View the node status of the running pod:

```
$ oc get node <node_name>   1
```

1   Specify the **<node_name>** that you obtained from the pod list.

Example output

```
NAME                     STATUS   ROLES         AGE   VERSION
ip-10-0-217-226.ec2.internal   Ready    infra,worker   17h   v1.25.0
```

Because the role list includes **infra**, the pod is running on the correct node.

6.9.2. Moving the default registry

You configure the registry Operator to deploy its pods to different nodes.

Prerequisites

- Configure additional compute machine sets in your OpenShift Container Platform cluster.

Procedure

1. View the **config/instance** object:

```
$ oc get configs.imageregistry.operator.openshift.io/cluster -o yaml
```

Example output

```
apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
 creationTimestamp: 2019-02-05T13:52:05Z
 finalizers:
 - imageregistry.operator.openshift.io/finalizer
 generation: 1
 name: cluster
 resourceVersion: "56174"
```

```
  selfLink: /apis/imageregistry.operator.openshift.io/v1/configs/cluster
  uid: 36fd3724-294d-11e9-a524-12ffeee2931b
spec:
  httpSecret: d9a012ccd117b1e6616ceccb2c3bb66a5fed1b5e481623
  logging: 2
  managementState: Managed
  proxy: {}
  replicas: 1
  requests:
    read: {}
    write: {}
  storage:
    s3:
      bucket: image-registry-us-east-1-c92e88cad85b48ec8b312344dff03c82-392c
      region: us-east-1
status:
  ...
```

2. Edit the **config/instance** object:

```
$ oc edit configs.imageregistry.operator.openshift.io/cluster
```

```
spec:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - podAffinityTerm:
          namespaces:
          - openshift-image-registry
          topologyKey: kubernetes.io/hostname
        weight: 100
  logLevel: Normal
  managementState: Managed
  nodeSelector: ❶
    node-role.kubernetes.io/infra: ""
  tolerations:
  - effect: NoSchedule
    key: node-role.kubernetes.io/infra
    value: reserved
  - effect: NoExecute
    key: node-role.kubernetes.io/infra
    value: reserved
```

❶ Add a **nodeSelector** parameter with the appropriate value to the component you want to move. You can use a **nodeSelector** in the format shown or use **<key>: <value>** pairs, based on the value specified for the node. If you added a taint to the infrasructure node, also add a matching toleration.

3. Verify the registry pod has been moved to the infrastructure node.

   a. Run the following command to identify the node where the registry pod is located:

   ```
   $ oc get pods -o wide -n openshift-image-registry
   ```

b. Confirm the node has the label you specified:

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

Review the command output and confirm that **node-role.kubernetes.io/infra** is in the **LABELS** list.

6.9.3. Moving the monitoring solution

The monitoring stack includes multiple components, including Prometheus, Thanos Querier, and Alertmanager. The Cluster Monitoring Operator manages this stack. To redeploy the monitoring stack to infrastructure nodes, you can create and apply a custom config map.

Procedure

1. Edit the **cluster-monitoring-config** config map and change the **nodeSelector** to use the **infra** label:

```
$ oc edit configmap cluster-monitoring-config -n openshift-monitoring
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |+
    alertmanagerMain:
      nodeSelector: ❶
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoExecute
    prometheusK8s:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoExecute
    prometheusOperator:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
```

```
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoExecute
    k8sPrometheusAdapter:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoExecute
    kubeStateMetrics:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoExecute
    telemeterClient:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoExecute
    openshiftStateMetrics:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoExecute
    thanosQuerier:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoExecute
```

1 Add a **nodeSelector** parameter with the appropriate value to the component you want to move. You can use a **nodeSelector** in the format shown or use **<key>: <value>** pairs,

2. Watch the monitoring pods move to the new machines:

```
$ watch 'oc get pod -n openshift-monitoring -o wide'
```

3. If a component has not moved to the **infra** node, delete the pod with this component:

```
$ oc delete pod -n openshift-monitoring <pod>
```

The component from the deleted pod is re-created on the **infra** node.

6.9.4. Moving logging resources

For information about moving logging resources, see:

- [Using node selectors to move logging resources](#)

- [Using taints and tolerations to control logging pod placement](#)

6.10. ABOUT THE CLUSTER AUTOSCALER

The cluster autoscaler adjusts the size of an OpenShift Container Platform cluster to meet its current deployment needs. It uses declarative, Kubernetes-style arguments to provide infrastructure management that does not rely on objects of a specific cloud provider. The cluster autoscaler has a cluster scope, and is not associated with a particular namespace.

The cluster autoscaler increases the size of the cluster when there are pods that fail to schedule on any of the current worker nodes due to insufficient resources or when another node is necessary to meet deployment needs. The cluster autoscaler does not increase the cluster resources beyond the limits that you specify.

The cluster autoscaler computes the total memory, CPU, and GPU on all nodes the cluster, even though it does not manage the control plane nodes. These values are not single-machine oriented. They are an aggregation of all the resources in the entire cluster. For example, if you set the maximum memory resource limit, the cluster autoscaler includes all the nodes in the cluster when calculating the current memory usage. That calculation is then used to determine if the cluster autoscaler has the capacity to add more worker resources.

> **IMPORTANT**
>
> Ensure that the **maxNodesTotal** value in the **ClusterAutoscaler** resource definition that you create is large enough to account for the total possible number of machines in your cluster. This value must encompass the number of control plane machines and the possible number of compute machines that you might scale to.

Every 10 seconds, the cluster autoscaler checks which nodes are unnecessary in the cluster and removes them. The cluster autoscaler considers a node for removal if the following conditions apply:

- The node utilization is less than the *node utilization level* threshold for the cluster. The node utilization level is the sum of the requested resources divided by the allocated resources for the node. If you do not specify a value in the **ClusterAutoscaler** custom resource, the cluster

autoscaler uses a default value of **0.5**, which corresponds to 50% utilization.

- The cluster autoscaler can move all pods running on the node to the other nodes. The Kubernetes scheduler is responsible for scheduling pods on the nodes.

- The cluster autoscaler does not have scale down disabled annotation.

If the following types of pods are present on a node, the cluster autoscaler will not remove the node:

- Pods with restrictive pod disruption budgets (PDBs).

- Kube-system pods that do not run on the node by default.

- Kube-system pods that do not have a PDB or have a PDB that is too restrictive.

- Pods that are not backed by a controller object such as a deployment, replica set, or stateful set.

- Pods with local storage.

- Pods that cannot be moved elsewhere because of a lack of resources, incompatible node selectors or affinity, matching anti-affinity, and so on.

- Unless they also have a **"cluster-autoscaler.kubernetes.io/safe-to-evict": "true"** annotation, pods that have a **"cluster-autoscaler.kubernetes.io/safe-to-evict": "false"** annotation.

For example, you set the maximum CPU limit to 64 cores and configure the cluster autoscaler to only create machines that have 8 cores each. If your cluster starts with 30 cores, the cluster autoscaler can add up to 4 more nodes with 32 cores, for a total of 62.

If you configure the cluster autoscaler, additional usage restrictions apply:

- Do not modify the nodes that are in autoscaled node groups directly. All nodes within the same node group have the same capacity and labels and run the same system pods.

- Specify requests for your pods.

- If you have to prevent pods from being deleted too quickly, configure appropriate PDBs.

- Confirm that your cloud provider quota is large enough to support the maximum node pools that you configure.

- Do not run additional node group autoscalers, especially the ones offered by your cloud provider.

The horizontal pod autoscaler (HPA) and the cluster autoscaler modify cluster resources in different ways. The HPA changes the deployment's or replica set's number of replicas based on the current CPU load. If the load increases, the HPA creates new replicas, regardless of the amount of resources available to the cluster. If there are not enough resources, the cluster autoscaler adds resources so that the HPA-created pods can run. If the load decreases, the HPA stops some replicas. If this action causes some nodes to be underutilized or completely empty, the cluster autoscaler deletes the unnecessary nodes.

The cluster autoscaler takes pod priorities into account. The Pod Priority and Preemption feature enables scheduling pods based on priorities if the cluster does not have enough resources, but the cluster autoscaler ensures that the cluster has resources to run all pods. To honor the intention of both features, the cluster autoscaler includes a priority cutoff function. You can use this cutoff to schedule "best-effort" pods, which do not cause the cluster autoscaler to increase resources but instead run only when spare resources are available.

Pods with priority lower than the cutoff value do not cause the cluster to scale up or prevent the cluster from scaling down. No new nodes are added to run the pods, and nodes running these pods might be deleted to free resources.

Cluster autoscaling is supported for the platforms that have machine API available on it.

6.10.1. Cluster autoscaler resource definition

This **ClusterAutoscaler** resource definition shows the parameters and sample values for the cluster autoscaler.

```
apiVersion: "autoscaling.openshift.io/v1"
kind: "ClusterAutoscaler"
metadata:
  name: "default"
spec:
  podPriorityThreshold: -10 1
  resourceLimits:
    maxNodesTotal: 24 2
    cores:
      min: 8 3
      max: 128 4
    memory:
      min: 4 5
      max: 256 6
    gpus:
      - type: nvidia.com/gpu 7
        min: 0 8
        max: 16 9
      - type: amd.com/gpu
        min: 0
        max: 4
  logVerbosity: 4 10
  scaleDown: 11
    enabled: true 12
    delayAfterAdd: 10m 13
    delayAfterDelete: 5m 14
    delayAfterFailure: 30s 15
    unneededTime: 5m 16
    utilizationThreshold: "0.4" 17
```

1 Specify the priority that a pod must exceed to cause the cluster autoscaler to deploy additional nodes. Enter a 32-bit integer value. The **podPriorityThreshold** value is compared to the value of the **PriorityClass** that you assign to each pod.

2 Specify the maximum number of nodes to deploy. This value is the total number of machines that are deployed in your cluster, not just the ones that the autoscaler controls. Ensure that this value is large enough to account for all of your control plane and compute machines and the total number of replicas that you specify in your **MachineAutoscaler** resources.

3 Specify the minimum number of cores to deploy in the cluster.

4 Specify the maximum number of cores to deploy in the cluster.

5 Specify the minimum amount of memory, in GiB, in the cluster.

6 Specify the maximum amount of memory, in GiB, in the cluster.

7 Optional: Specify the type of GPU node to deploy. Only **nvidia.com/gpu** and **amd.com/gpu** are valid types.

8 Specify the minimum number of GPUs to deploy in the cluster.

9 Specify the maximum number of GPUs to deploy in the cluster.

10 Specify the logging verbosity level between **0** and **10**. The following log level thresholds are provided for guidance:

- **1**: (Default) Basic information about changes.

- **4**: Debug-level verbosity for troubleshooting typical issues.

- **9**: Extensive, protocol-level debugging information.

If you do not specify a value, the default value of **1** is used.

11 In this section, you can specify the period to wait for each action by using any valid [ParseDuration](#) interval, including **ns**, **us**, **ms**, **s**, **m**, and **h**.

12 Specify whether the cluster autoscaler can remove unnecessary nodes.

13 Optional: Specify the period to wait before deleting a node after a node has recently been *added*. If you do not specify a value, the default value of **10m** is used.

14 Optional: Specify the period to wait before deleting a node after a node has recently been *deleted*. If you do not specify a value, the default value of **0s** is used.

15 Optional: Specify the period to wait before deleting a node after a scale down failure occurred. If you do not specify a value, the default value of **3m** is used.

16 Optional: Specify a period of time before an unnecessary node is eligible for deletion. If you do not specify a value, the default value of **10m** is used.

17 Optional: Specify the *node utilization level*. Nodes below this utilization level are eligible for deletion.

The node utilization level is the sum of the requested resources divided by the allocated resources for the node, and must be a value greater than **"0"** but less than **"1"**. If you do not specify a value, the cluster autoscaler uses a default value of **"0.5"**, which corresponds to 50% utilization. You must express this value as a string.

> **NOTE**
>
> When performing a scaling operation, the cluster autoscaler remains within the ranges set in the **ClusterAutoscaler** resource definition, such as the minimum and maximum number of cores to deploy or the amount of memory in the cluster. However, the cluster autoscaler does not correct the current values in your cluster to be within those ranges.
>
> The minimum and maximum CPUs, memory, and GPU values are determined by calculating those resources on all nodes in the cluster, even if the cluster autoscaler does not manage the nodes. For example, the control plane nodes are considered in the total memory in the cluster, even though the cluster autoscaler does not manage the control plane nodes.

6.10.2. Deploying a cluster autoscaler

To deploy a cluster autoscaler, you create an instance of the **ClusterAutoscaler** resource.

Procedure

1. Create a YAML file for a **ClusterAutoscaler** resource that contains the custom resource definition.

2. Create the custom resource in the cluster by running the following command:

   ```
   $ oc create -f <filename>.yaml ❶
   ```

   ❶   **<filename>** is the name of the custom resource file.

6.11. ABOUT THE MACHINE AUTOSCALER

The machine autoscaler adjusts the number of Machines in the compute machine sets that you deploy in an OpenShift Container Platform cluster. You can scale both the default **worker** compute machine set and any other compute machine sets that you create. The machine autoscaler makes more Machines when the cluster runs out of resources to support more deployments. Any changes to the values in **MachineAutoscaler** resources, such as the minimum or maximum number of instances, are immediately applied to the compute machine set they target.

> **IMPORTANT**
>
> You must deploy a machine autoscaler for the cluster autoscaler to scale your machines. The cluster autoscaler uses the annotations on compute machine sets that the machine autoscaler sets to determine the resources that it can scale. If you define a cluster autoscaler without also defining machine autoscalers, the cluster autoscaler will never scale your cluster.

6.11.1. Machine autoscaler resource definition

This **MachineAutoscaler** resource definition shows the parameters and sample values for the machine autoscaler.

```
apiVersion: "autoscaling.openshift.io/v1beta1"
kind: "MachineAutoscaler"
metadata:
```

```
      name: "worker-us-east-1a" 1
      namespace: "openshift-machine-api"
   spec:
     minReplicas: 1 2
     maxReplicas: 12 3
     scaleTargetRef: 4
       apiVersion: machine.openshift.io/v1beta1
       kind: MachineSet 5
       name: worker-us-east-1a 6
```

1 Specify the machine autoscaler name. To make it easier to identify which compute machine set this machine autoscaler scales, specify or include the name of the compute machine set to scale. The compute machine set name takes the following form: **<clusterid>-<machineset>-<region>**.

2 Specify the minimum number machines of the specified type that must remain in the specified zone after the cluster autoscaler initiates cluster scaling. If running in AWS, GCP, Azure, RHOSP, or vSphere, this value can be set to **0**. For other providers, do not set this value to **0**.

You can save on costs by setting this value to **0** for use cases such as running expensive or limited-usage hardware that is used for specialized workloads, or by scaling a compute machine set with extra large machines. The cluster autoscaler scales the compute machine set down to zero if the machines are not in use.

> **IMPORTANT**
>
> Do not set the **spec.minReplicas** value to **0** for the three compute machine sets that are created during the OpenShift Container Platform installation process for an installer provisioned infrastructure.

3 Specify the maximum number machines of the specified type that the cluster autoscaler can deploy in the specified zone after it initiates cluster scaling. Ensure that the **maxNodesTotal** value in the **ClusterAutoscaler** resource definition is large enough to allow the machine autoscaler to deploy this number of machines.

4 In this section, provide values that describe the existing compute machine set to scale.

5 The **kind** parameter value is always **MachineSet**.

6 The **name** value must match the name of an existing compute machine set, as shown in the **metadata.name** parameter value.

6.11.2. Deploying a machine autoscaler

To deploy a machine autoscaler, you create an instance of the **MachineAutoscaler** resource.

Procedure

1. Create a YAML file for a **MachineAutoscaler** resource that contains the custom resource definition.

2. Create the custom resource in the cluster by running the following command:

   ```
   $ oc create -f <filename>.yaml 1
   ```

1 **<filename>** is the name of the custom resource file.

6.12. CONFIGURING LINUX CGROUP V2

You can enable Linux control group version 2 (cgroup v2) in your cluster by editing the **node.config** object. Enabling cgroup v2 in OpenShift Container Platform disables all cgroups version 1 controllers and hierarchies in your cluster. cgroup v1 is enabled by default.

cgroup v2 is the current version of the Linux cgroup API. cgroup v2 offers several improvements over cgroup v1, including a unified hierarchy, safer sub-tree delegation, new features such as Pressure Stall Information, and enhanced resource management and isolation. However, cgroup v2 has different CPU, memory, and I/O management characteristics than cgroup v1. Therefore, some workloads might experience slight differences in memory or CPU usage on clusters that run cgroup v2.
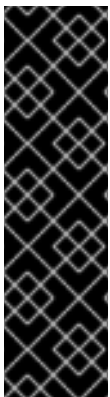
> **IMPORTANT**
>
> OpenShift Container Platform cgroups version 2 support is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. 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 .

> **NOTE**
>
> Currently, disabling CPU load balancing is not supported by cgroup v2. As a result, you might not get the desired behavior from performance profiles if you have cgroup v2 enabled. Enabling cgroup v2 is not recommended if you are using performance profiles.

Prerequisites

- You have a running OpenShift Container Platform cluster that uses version 4.12 or later.

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

- You have enabled the **TechPreviewNoUpgrade** feature set by using the feature gates.

Procedure

1. Enable cgroup v2 on nodes:

   a. Edit the **node.config** object:

      ```
      $ oc edit nodes.config/cluster
      ```

   b. Add **spec.cgroupMode: "v2"**:

      Example node.config object

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

```
kind: Node
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2022-07-08T16:02:51Z"
  generation: 1
  name: cluster
  ownerReferences:
  - apiVersion: config.openshift.io/v1
    kind: ClusterVersion
    name: version
    uid: 36282574-bf9f-409e-a6cd-3032939293eb
  resourceVersion: "1865"
  uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
  cgroupMode: "v2"  ❶
...
```

❶   Enables cgroup v2.

Verification

1. Check the machine configs to see that the new machine configs were added:

```
$ oc get mc
```

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
97-master-generated-kubelet                 52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0        3m  ❶
99-worker-generated-kubelet                 52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0        3m
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
worker-enable-cgroups-v2                                       3.2.0        10s
```

1 New machine configs are created, as expected.

2. Check that the new **kernelArguments** were added to the new machine configs:

   ```
   $ oc describe mc <name>
   ```

   Example output

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: worker
     name: 05-worker-kernelarg-selinuxpermissive
   spec:
     kernelArguments:
     - systemd_unified_cgroup_hierarchy=1  1
     - cgroup_no_v1="all"  2
     - psi=1  3
   ```

   1 Enables cgroup v2 in systemd.

   2 Disables cgroups v1.

   3 Enables the Linux Pressure Stall Information (PSI) feature.

3. 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
   ci-ln-fm1qnwt-72292-99kt6-master-0      Ready                   master  58m  v1.25.0
   ci-ln-fm1qnwt-72292-99kt6-master-1      Ready                   master  58m  v1.25.0
   ci-ln-fm1qnwt-72292-99kt6-master-2      Ready                   master  58m  v1.25.0
   ci-ln-fm1qnwt-72292-99kt6-worker-a-h5gt4   Ready,SchedulingDisabled  worker  48m
   v1.25.0
   ci-ln-fm1qnwt-72292-99kt6-worker-b-7vtmd   Ready                   worker  48m  v1.25.0
   ci-ln-fm1qnwt-72292-99kt6-worker-c-rhzkv   Ready                   worker  48m  v1.25.0
   ```

4. After a node returns to the **Ready** state, start a debug session for that node:

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

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

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

6. Check that the **sys/fs/cgroup/cgroup2fs** file is present on your nodes. This file is created by cgroup v2:

```
$ stat -c %T -f /sys/fs/cgroup
```

Example output

```
cgroup2fs
```

6.13. ENABLING TECHNOLOGY PREVIEW FEATURES USING FEATUREGATES

You can turn on a subset of the current Technology Preview features on for all nodes in the cluster by editing the **FeatureGate** custom resource (CR).

6.13.1. Understanding feature gates

You can use the **FeatureGate** custom resource (CR) to enable specific feature sets in your cluster. A feature set is a collection of OpenShift Container Platform features that are not enabled by default.

You can activate the following feature set by using the **FeatureGate** CR:

- **TechPreviewNoUpgrade**. This feature set is a subset of the current Technology Preview features. This feature set allows you to enable these Technology Preview features on test clusters, where you can fully test them, while leaving the features disabled on production clusters.

  > **WARNING**
  >
  > Enabling the **TechPreviewNoUpgrade** feature set on your cluster cannot be undone and prevents minor version updates. You should not enable this feature set on production clusters.

  The following Technology Preview features are enabled by this feature set:

  - CSI automatic migration. Enables automatic migration for supported in-tree volume plugins to their equivalent Container Storage Interface (CSI) drivers. Supported for:
    - Azure File (**CSIMigrationAzureFile**)
    - VMware vSphere (**CSIMigrationvSphere**)

- Shared Resources CSI Driver and Build CSI Volumes in OpenShift Builds. Enables the Container Storage Interface (CSI). (**CSIDriverSharedResource**)

- CSI volumes. Enables CSI volume support for the OpenShift Container Platform build system. (**BuildCSIVolumes**)

- Swap memory on nodes. Enables swap memory use for OpenShift Container Platform workloads on a per-node basis. (**NodeSwap**)

- cgroups v2. Enables cgroup v2, the next version of the Linux cgroup API. (**CGroupsV2**)

- crun. Enables the crun container runtime. (**Crun**)

- Insights Operator. Enables the Insights Operator, which gathers OpenShift Container Platform configuration data and sends it to Red Hat. (**InsightsConfigAPI**)

- External cloud providers. Enables support for external cloud providers for clusters on vSphere, AWS, Azure, and GCP. Support for OpenStack is GA. (**ExternalCloudProvider**)

- Pod topology spread constraints. Enables the **matchLabelKeys** parameter for pod topology constraints. The parameter is list of pod label keys to select the pods over which spreading will be calculated. (**MatchLabelKeysInPodTopologySpread**)

- Pod security admission enforcement. Enables restricted enforcement for pod security admission. Instead of only logging a warning, pods are rejected if they violate pod security standards. (**OpenShiftPodSecurityAdmission**)

> **NOTE**
>
> Pod security admission restricted enforcement is only activated if you enable the **TechPreviewNoUpgrade** feature set after your OpenShift Container Platform cluster is installed. It is not activated if you enable the **TechPreviewNoUpgrade** feature set during cluster installation.

6.13.2. Enabling feature sets using the web console

You can use the OpenShift Container Platform web console to enable feature sets for all of the nodes in a cluster by editing the **FeatureGate** custom resource (CR).

Procedure

To enable feature sets:

1. In the OpenShift Container Platform web console, switch to the **Administration → Custom Resource Definitions** page.

2. On the **Custom Resource Definitions** page, click **FeatureGate**.

3. On the **Custom Resource Definition Details** page, click the **Instances** tab.

4. Click the **cluster** feature gate, then click the **YAML** tab.

5. Edit the **cluster** instance to add specific feature sets:

> **WARNING**
>
> Enabling the **TechPreviewNoUpgrade** feature set on your cluster cannot be undone and prevents minor version updates. You should not enable this feature set on production clusters.

Sample Feature Gate custom resource

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster ❶
# ...
spec:
  featureSet: TechPreviewNoUpgrade ❷
```

❶ The name of the **FeatureGate** CR must be **cluster**.

❷ Add the feature set that you want to enable:

- **TechPreviewNoUpgrade** enables specific Technology Preview features.

After you save the changes, new machine configs are created, the machine config pools are updated, and scheduling on each node is disabled while the change is being applied.

Verification

You can verify that the feature gates are enabled by looking at the **kubelet.conf** file on a node after the nodes return to the ready state.

1. From the **Administrator** perspective in the web console, navigate to **Compute → Nodes**.

2. Select a node.

3. In the **Node details** page, click **Terminal**.

4. In the terminal window, change your root directory to **/host**:

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

5. View the **kubelet.conf** file:

   ```
   sh-4.2# cat /etc/kubernetes/kubelet.conf
   ```

Sample output

```
# ...
featureGates:
  InsightsOperatorPullingSCA: true,
```

```
    LegacyNodeRoleBehavior: false
    # ...
```

The features that are listed as **true** are enabled on your cluster.

> **NOTE**
>
> The features listed vary depending upon the OpenShift Container Platform version.

6.13.3. Enabling feature sets using the CLI

You can use the OpenShift CLI (**oc**) to enable feature sets for all of the nodes in a cluster by editing the **FeatureGate** custom resource (CR).

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

To enable feature sets:

1. Edit the **FeatureGate** CR named **cluster**:

   ```
   $ oc edit featuregate cluster
   ```

   > **WARNING**
   >
   > Enabling the **TechPreviewNoUpgrade** feature set on your cluster cannot be undone and prevents minor version updates. You should not enable this feature set on production clusters.

   Sample FeatureGate custom resource

   ```
   apiVersion: config.openshift.io/v1
   kind: FeatureGate
   metadata:
     name: cluster 1
   # ...
   spec:
     featureSet: TechPreviewNoUpgrade 2
   ```

   1 The name of the **FeatureGate** CR must be **cluster**.

   2 Add the feature set that you want to enable:

   - **TechPreviewNoUpgrade** enables specific Technology Preview features.

After you save the changes, new machine configs are created, the machine config pools are updated, and scheduling on each node is disabled while the change is being applied.

Verification

You can verify that the feature gates are enabled by looking at the **kubelet.conf** file on a node after the nodes return to the ready state.

1. From the **Administrator** perspective in the web console, navigate to **Compute → Nodes**.

2. Select a node.

3. In the **Node details** page, click **Terminal**.

4. In the terminal window, change your root directory to **/host**:

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

5. View the **kubelet.conf** file:

   ```
   sh-4.2# cat /etc/kubernetes/kubelet.conf
   ```

Sample output

```
# ...
featureGates:
  InsightsOperatorPullingSCA: true,
  LegacyNodeRoleBehavior: false
# ...
```

The features that are listed as **true** are enabled on your cluster.

> **NOTE**
>
> The features listed vary depending upon the OpenShift Container Platform version.

6.14. ETCD TASKS

Back up etcd, enable or disable etcd encryption, or defragment etcd data.

6.14.1. About etcd encryption

By default, etcd data is not encrypted in OpenShift Container Platform. You can enable etcd encryption for your cluster to provide an additional layer of data security. For example, it can help protect the loss of sensitive data if an etcd backup is exposed to the incorrect parties.

When you enable etcd encryption, the following OpenShift API server and Kubernetes API server resources are encrypted:

- Secrets

- Config maps

- Routes

- OAuth access tokens

- OAuth authorize tokens

When you enable etcd encryption, encryption keys are created. These keys are rotated on a weekly basis. You must have these keys to restore from an etcd backup.

> **NOTE**
>
> Etcd encryption only encrypts values, not keys. Resource types, namespaces, and object names are unencrypted.
>
> If etcd encryption is enabled during a backup, the ***static_kuberesources_<datetimestamp>.tar.gz*** file contains the encryption keys for the etcd snapshot. For security reasons, store this file separately from the etcd snapshot. However, this file is required to restore a previous state of etcd from the respective etcd snapshot.

6.14.2. Enabling etcd encryption

You can enable etcd encryption to encrypt sensitive resources in your cluster.

> **WARNING**
>
> Do not back up etcd resources until the initial encryption process is completed. If the encryption process is not completed, the backup might be only partially encrypted.
>
> After you enable etcd encryption, several changes can occur:
>
> - The etcd encryption might affect the memory consumption of a few resources.
>
> - You might notice a transient affect on backup performance because the leader must serve the backup.
>
> - A disk I/O can affect the node that receives the backup state.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Modify the **APIServer** object:

   ```
   $ oc edit apiserver
   ```

2. Set the **encryption** field type to **aescbc**:

```
spec:
  encryption:
    type: aescbc
```
1

1   The **aescbc** type means that AES-CBC with PKCS#7 padding and a 32 byte key is used to perform the encryption.

3. Save the file to apply the changes.
   The encryption process starts. It can take 20 minutes or longer for this process to complete, depending on the size of your cluster.

4. Verify that etcd encryption was successful.

   a. Review the **Encrypted** status condition for the OpenShift API server to verify that its resources were successfully encrypted:

   ```
   $ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?
   (@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
   ```

   The output shows **EncryptionCompleted** upon successful encryption:

   ```
   EncryptionCompleted
   All resources encrypted: routes.route.openshift.io
   ```

   If the output shows **EncryptionInProgress**, encryption is still in progress. Wait a few minutes and try again.

   b. Review the **Encrypted** status condition for the Kubernetes API server to verify that its resources were successfully encrypted:

   ```
   $ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
   (@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
   ```

   The output shows **EncryptionCompleted** upon successful encryption:

   ```
   EncryptionCompleted
   All resources encrypted: secrets, configmaps
   ```

   If the output shows **EncryptionInProgress**, encryption is still in progress. Wait a few minutes and try again.

   c. Review the **Encrypted** status condition for the OpenShift OAuth API server to verify that its resources were successfully encrypted:

   ```
   $ oc get authentication.operator.openshift.io -o=jsonpath='{range
   .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
   ```

   The output shows **EncryptionCompleted** upon successful encryption:

> EncryptionCompleted
> All resources encrypted: oauthaccesstokens.oauth.openshift.io,
> oauthauthorizetokens.oauth.openshift.io

If the output shows **EncryptionInProgress**, encryption is still in progress. Wait a few
minutes and try again.

6.14.3. Disabling etcd encryption

You can disable encryption of etcd data in your cluster.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Modify the **APIServer** object:

   > $ oc edit apiserver

2. Set the **encryption** field type to **identity**:

   > spec:
   >   encryption:
   >   type: identity **1**

   1   The **identity** type is the default value and means that no encryption is performed.

3. Save the file to apply the changes.
   The decryption process starts. It can take 20 minutes or longer for this process to complete,
   depending on the size of your cluster.

4. Verify that etcd decryption was successful.

   a. Review the **Encrypted** status condition for the OpenShift API server to verify that its
      resources were successfully decrypted:

      > $ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?
      > (@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'

      The output shows **DecryptionCompleted** upon successful decryption:

      > DecryptionCompleted
      > Encryption mode set to identity and everything is decrypted

      If the output shows **DecryptionInProgress**, decryption is still in progress. Wait a few
      minutes and try again.

   b. Review the **Encrypted** status condition for the Kubernetes API server to verify that its
      resources were successfully decrypted:

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

The output shows **DecryptionCompleted** upon successful decryption:

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

If the output shows **DecryptionInProgress**, decryption is still in progress. Wait a few minutes and try again.

c. Review the **Encrypted** status condition for the OpenShift OAuth API server to verify that its resources were successfully decrypted:

```
$ oc get authentication.operator.openshift.io -o=jsonpath='{range
.items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

The output shows **DecryptionCompleted** upon successful decryption:

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

If the output shows **DecryptionInProgress**, decryption is still in progress. Wait a few minutes and try again.

6.14.4. Backing up etcd data

Follow these steps to back up etcd data by creating an etcd snapshot and backing up the resources for the static pods. This backup can be saved and used at a later time if you need to restore etcd.

IMPORTANT

Only save a backup from a single control plane host. Do not take a backup from each control plane host in the cluster.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

- You have checked whether the cluster-wide proxy is enabled.

  ### TIP

  You can check whether the proxy is enabled by reviewing the output of **oc get proxy cluster -o yaml**. The proxy is enabled if the **httpProxy**, **httpsProxy**, and **noProxy** fields have values set.

Procedure

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

   ```
   $ oc debug --as-root node/<node_name>
   ```

2. Change your root directory to /**host** in the debug shell:

> sh-4.4# chroot /host

3. If the cluster-wide proxy is enabled, be sure that you have exported the **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** environment variables.

4. Run the **cluster-backup.sh** script in the debug shell and pass in the location to save the backup to.

TIP

The **cluster-backup.sh** script is maintained as a component of the etcd Cluster Operator and is a wrapper around the **etcdctl snapshot save** command.

> sh-4.4# /usr/local/bin/cluster-backup.sh /home/core/assets/backup

Example script output

> found latest kube-apiserver: /etc/kubernetes/static-pod-resources/kube-apiserver-pod-6
> found latest kube-controller-manager: /etc/kubernetes/static-pod-resources/kube-controller-manager-pod-7
> found latest kube-scheduler: /etc/kubernetes/static-pod-resources/kube-scheduler-pod-6
> found latest etcd: /etc/kubernetes/static-pod-resources/etcd-pod-3
> ede95fe6b88b87ba86a03c15e669fb4aa5bf0991c180d3c6895ce72eaade54a1
> etcdctl version: 3.4.14
> API version: 3.4
> {"level":"info","ts":1624647639.0188997,"caller":"snapshot/v3_snapshot.go:119","msg":"created temporary db file","path":"/home/core/assets/backup/snapshot_2021-06-25_190035.db.part"}
> {"level":"info","ts":"2021-06-25T19:00:39.030Z","caller":"clientv3/maintenance.go:200","msg":"opened snapshot stream; downloading"}
> {"level":"info","ts":1624647639.0301006,"caller":"snapshot/v3_snapshot.go:127","msg":"fetching snapshot","endpoint":"https://10.0.0.5:2379"}
> {"level":"info","ts":"2021-06-25T19:00:40.215Z","caller":"clientv3/maintenance.go:208","msg":"completed snapshot read; closing"}
> {"level":"info","ts":1624647640.6032252,"caller":"snapshot/v3_snapshot.go:142","msg":"fetched snapshot","endpoint":"https://10.0.0.5:2379","size":"114 MB","took":1.584090459}
> {"level":"info","ts":1624647640.6047094,"caller":"snapshot/v3_snapshot.go:152","msg":"saved", "path":"/home/core/assets/backup/snapshot_2021-06-25_190035.db"}
> Snapshot saved at /home/core/assets/backup/snapshot_2021-06-25_190035.db
> {"hash":3866667823,"revision":31407,"totalKey":12828,"totalSize":114446336}
> snapshot db and kube resources are successfully saved to /home/core/assets/backup

In this example, two files are created in the **/home/core/assets/backup/** directory on the control plane host:

- **snapshot_<datetimestamp>.db**: This file is the etcd snapshot. The **cluster-backup.sh** script confirms its validity.

- **static_kuberesources_<datetimestamp>.tar.gz**: This file contains the resources for the static pods. If etcd encryption is enabled, it also contains the encryption keys for the etcd snapshot.

> **NOTE**
>
> If etcd encryption is enabled, it is recommended to store this second file separately from the etcd snapshot for security reasons. However, this file is required to restore from the etcd snapshot.
>
> Keep in mind that etcd encryption only encrypts values, not keys. This means that resource types, namespaces, and object names are unencrypted.

6.14.5. Defragmenting etcd data

For large and dense clusters, etcd can suffer from poor performance if the keyspace grows too large and exceeds the space quota. Periodically maintain and defragment etcd to free up space in the data store. Monitor Prometheus for etcd metrics and defragment it when required; otherwise, etcd can raise a cluster-wide alarm that puts the cluster into a maintenance mode that accepts only key reads and deletes.

Monitor these key metrics:

- **etcd_server_quota_backend_bytes**, which is the current quota limit

- **etcd_mvcc_db_total_size_in_use_in_bytes**, which indicates the actual database usage after a history compaction

- **etcd_mvcc_db_total_size_in_bytes**, which shows the database size, including free space waiting for defragmentation

Defragment etcd data to reclaim disk space after events that cause disk fragmentation, such as etcd history compaction.

History compaction is performed automatically every five minutes and leaves gaps in the back-end database. This fragmented space is available for use by etcd, but is not available to the host file system. You must defragment etcd to make this space available to the host file system.

Defragmentation occurs automatically, but you can also trigger it manually.

> **NOTE**
>
> Automatic defragmentation is good for most cases, because the etcd operator uses cluster information to determine the most efficient operation for the user.

6.14.5.1. Automatic defragmentation

The etcd Operator automatically defragments disks. No manual intervention is needed.

Verify that the defragmentation process is successful by viewing one of these logs:

- etcd logs

- cluster-etcd-operator pod

- operator status error log

> **WARNING**
>
> Automatic defragmentation can cause leader election failure in various OpenShift core components, such as the Kubernetes controller manager, which triggers a restart of the failing component. The restart is harmless and either triggers failover to the next running instance or the component resumes work again after the restart.

Example log output for successful defragmentation

> etcd member has been defragmented: *<member_name>*, memberID: *<member_id>*

Example log output for unsuccessful defragmentation

> failed defrag on member: *<member_name>*, memberID: *<member_id>*: *<error_message>*

6.14.5.2. Manual defragmentation

A Prometheus alert indicates when you need to use manual defragmentation. The alert is displayed in two cases:

- When etcd uses more than 50% of its available space for more than 10 minutes

- When etcd is actively using less than 50% of its total database size for more than 10 minutes

You can also determine whether defragmentation is needed by checking the etcd database size in MB that will be freed by defragmentation with the PromQL expression: **(etcd_mvcc_db_total_size_in_bytes - etcd_mvcc_db_total_size_in_use_in_bytes)/1024/1024**

> **WARNING**
>
> Defragmenting etcd is a blocking action. The etcd member will not respond until defragmentation is complete. For this reason, wait at least one minute between defragmentation actions on each of the pods to allow the cluster to recover.

Follow this procedure to defragment etcd data on each etcd member.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Determine which etcd member is the leader, because the leader should be defragmented last.

a. Get the list of etcd pods:

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd -o wide
```

Example output

```
etcd-ip-10-0-159-225.example.redhat.com                3/3    Running   0       175m
10.0.159.225   ip-10-0-159-225.example.redhat.com   <none>         <none>
etcd-ip-10-0-191-37.example.redhat.com                 3/3    Running   0       173m
10.0.191.37    ip-10-0-191-37.example.redhat.com    <none>         <none>
etcd-ip-10-0-199-170.example.redhat.com                3/3    Running   0       176m
10.0.199.170   ip-10-0-199-170.example.redhat.com   <none>         <none>
```

b. Choose a pod and run the following command to determine which etcd member is the leader:

```
$ oc rsh -n openshift-etcd etcd-ip-10-0-159-225.example.redhat.com etcdctl endpoint
status --cluster -w table
```

Example output

```
Defaulting container name to etcdctl.
Use 'oc describe pod/etcd-ip-10-0-159-225.example.redhat.com -n openshift-etcd' to see
all of the containers in this pod.
+-------------------------+------------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
|       ENDPOINT          |      ID      | VERSION | DB SIZE | IS LEADER | IS LEARNER |
RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+-------------------------+------------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
|  https://10.0.191.37:2379 | 251cd44483d811c3 |  3.4.9 |  104 MB |   false |    false |
7 |    91624 |          91624 |      |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |  3.4.9 |  104 MB |   false |    false |
7 |    91624 |          91624 |      |
| https://10.0.199.170:2379 | 9ac311f93915cc79 |  3.4.9 |  104 MB |    true |    false |
7 |    91624 |          91624 |      |
+-------------------------+------------------+---------+---------+-----------+------------+-----------
+------------+-------------------+--------+
```

Based on the **IS LEADER** column of this output, the **https://10.0.199.170:2379** endpoint is the leader. Matching this endpoint with the output of the previous step, the pod name of the leader is **etcd-ip-10-0-199-170.example.redhat.com**.

2. Defragment an etcd member.

a. Connect to the running etcd container, passing in the name of a pod that is *not* the leader:

```
$ oc rsh -n openshift-etcd etcd-ip-10-0-159-225.example.redhat.com
```

b. Unset the **ETCDCTL_ENDPOINTS** environment variable:

```
sh-4.4# unset ETCDCTL_ENDPOINTS
```

c. Defragment the etcd member:

```
sh-4.4# etcdctl --command-timeout=30s --endpoints=https://localhost:2379 defrag
```

Example output

```
Finished defragmenting etcd member[https://localhost:2379]
```

If a timeout error occurs, increase the value for **--command-timeout** until the command succeeds.

d. Verify that the database size was reduced:

```
sh-4.4# etcdctl endpoint status -w table --cluster
```

Example output

```
+-------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
|        ENDPOINT         |       ID         | VERSION | DB SIZE | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+-------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
|  https://10.0.191.37:2379 | 251cd44483d811c3 |  3.4.9 | 104 MB |    false |    false |
7 |   91624 |          91624 |      |
| https://10.0.159.225:2379 | 264c7c58ecbdabee |  3.4.9 |  41 MB |    false |    false |
7 |   91624 |          91624 |      | ❶
| https://10.0.199.170:2379 | 9ac311f93915cc79 |  3.4.9 | 104 MB |     true |    false |
7 |   91624 |          91624 |      |
+-------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
```

This example shows that the database size for this etcd member is now 41 MB as opposed to the starting size of 104 MB.

e. Repeat these steps to connect to each of the other etcd members and defragment them. Always defragment the leader last.
Wait at least one minute between defragmentation actions to allow the etcd pod to recover. Until the etcd pod recovers, the etcd member will not respond.

3. If any **NOSPACE** alarms were triggered due to the space quota being exceeded, clear them.

a. Check if there are any **NOSPACE** alarms:

```
sh-4.4# etcdctl alarm list
```

Example output

```
memberID:12345678912345678912 alarm:NOSPACE
```

b. Clear the alarms:

```
sh-4.4# etcdctl alarm disarm
```

6.14.6. Restoring to a previous cluster state

You can use a saved etcd backup to restore a previous cluster state or restore a cluster that has lost the majority of control plane hosts.

> **NOTE**
>
> If your cluster uses a control plane machine set, see "Troubleshooting the control plane machine set" for a more simple etcd recovery procedure.

> **IMPORTANT**
>
> When you restore your cluster, you must use an etcd backup that was taken from the same z-stream release. For example, an OpenShift Container Platform 4.7.2 cluster must use an etcd backup that was taken from 4.7.2.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role through a certificate-based **kubeconfig** file, like the one that was used during installation.

- A healthy control plane host to use as the recovery host.

- SSH access to control plane hosts.

- A backup directory containing both the etcd snapshot and the resources for the static pods, which were from the same backup. The file names in the directory must be in the following formats: **snapshot_<datetimestamp>.db** and **static_kuberesources_<datetimestamp>.tar.gz**.

> **IMPORTANT**
>
> For non-recovery control plane nodes, it is not required to establish SSH connectivity or to stop the static pods. You can delete and recreate other non-recovery, control plane machines, one by one.

Procedure

1. Select a control plane host to use as the recovery host. This is the host that you will run the restore operation on.

2. Establish SSH connectivity to each of the control plane nodes, including the recovery host. The Kubernetes API server becomes inaccessible after the restore process starts, so you cannot access the control plane nodes. For this reason, it is recommended to establish SSH connectivity to each control plane host in a separate terminal.

   > **IMPORTANT**
   >
   > If you do not complete this step, you will not be able to access the control plane hosts to complete the restore procedure, and you will be unable to recover your cluster from this state.

3. Copy the etcd backup directory to the recovery control plane host.

This procedure assumes that you copied the **backup** directory containing the etcd snapshot and the resources for the static pods to the **/home/core/** directory of your recovery control plane host.

4. Stop the static pods on any other control plane nodes.

> **NOTE**
>
> You do not need to stop the static pods on the recovery host.

a. Access a control plane host that is not the recovery host.

b. Move the existing etcd pod file out of the kubelet manifest directory:

```
$ sudo mv -v /etc/kubernetes/manifests/etcd-pod.yaml /tmp
```

c. Verify that the etcd pods are stopped.

```
$ sudo crictl ps | grep etcd | egrep -v "operator|etcd-guard"
```

The output of this command should be empty. If it is not empty, wait a few minutes and check again.

d. Move the existing Kubernetes API server pod file out of the kubelet manifest directory:

```
$ sudo mv -v /etc/kubernetes/manifests/kube-apiserver-pod.yaml /tmp
```

e. Verify that the Kubernetes API server pods are stopped.

```
$ sudo crictl ps | grep kube-apiserver | egrep -v "operator|guard"
```

The output of this command should be empty. If it is not empty, wait a few minutes and check again.

f. Move the etcd data directory to a different location:

```
$ sudo mv -v /var/lib/etcd/ /tmp
```

g. If the **/etc/kubernetes/manifests/keepalived.yaml** file exists and the node is deleted, follow these steps:

i. Move the **/etc/kubernetes/manifests/keepalived.yaml** file out of the kubelet manifest directory:

```
$ sudo mv -v /etc/kubernetes/manifests/keepalived.yaml /tmp
```

ii. Verify that any containers managed by the **keepalived** daemon are stopped:

```
$ sudo crictl ps --name keepalived
```

The output of this command should be empty. If it is not empty, wait a few minutes and check again.

      iii. Check if the control plane has any Virtual IPs (VIPs) assigned to it:

```
$ ip -o address | egrep '<api_vip>|<ingress_vip>'
```

      iv. For each reported VIP, run the following command to remove it:

```
$ sudo ip address del <reported_vip> dev <reported_vip_device>
```

   h. Repeat this step on each of the other control plane hosts that is not the recovery host.

5. Access the recovery control plane host.

6. If the **keepalived** daemon is in use, verify that the recovery control plane node owns the VIP:

```
$ ip -o address | grep <api_vip>
```

The address of the VIP is highlighted in the output if it exists. This command returns an empty string if the VIP is not set or configured incorrectly.

7. If the cluster-wide proxy is enabled, be sure that you have exported the **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** environment variables.

TIP

You can check whether the proxy is enabled by reviewing the output of **oc get proxy cluster -o yaml**. The proxy is enabled if the **httpProxy**, **httpsProxy**, and **noProxy** fields have values set.

8. Run the restore script on the recovery control plane host and pass in the path to the etcd backup directory:

```
$ sudo -E /usr/local/bin/cluster-restore.sh /home/core/assets/backup
```

Example script output

```
...stopping kube-scheduler-pod.yaml
...stopping kube-controller-manager-pod.yaml
...stopping etcd-pod.yaml
...stopping kube-apiserver-pod.yaml
Waiting for container etcd to stop
.complete
Waiting for container etcdctl to stop
...........................complete
Waiting for container etcd-metrics to stop
complete
Waiting for container kube-controller-manager to stop
complete
Waiting for container kube-apiserver to stop
............................................................................complete
Waiting for container kube-scheduler to stop
complete
Moving etcd data-dir /var/lib/etcd/member to /var/lib/etcd-backup
starting restore-etcd static pod
starting kube-apiserver-pod.yaml
static-pod-resources/kube-apiserver-pod-7/kube-apiserver-pod.yaml
```

> starting kube-controller-manager-pod.yaml
> static-pod-resources/kube-controller-manager-pod-7/kube-controller-manager-pod.yaml
> starting kube-scheduler-pod.yaml
> static-pod-resources/kube-scheduler-pod-8/kube-scheduler-pod.yaml

> **NOTE**
>
> The restore process can cause nodes to enter the **NotReady** state if the node certificates were updated after the last etcd backup.

9. Check the nodes to ensure they are in the **Ready** state.

   a. Run the following command:

      ```
      $ oc get nodes -w
      ```

      Sample output

      ```
      NAME              STATUS  ROLES        AGE    VERSION
      host-172-25-75-28  Ready   master       3d20h  v1.25.0
      host-172-25-75-38  Ready   infra,worker 3d20h  v1.25.0
      host-172-25-75-40  Ready   master       3d20h  v1.25.0
      host-172-25-75-65  Ready   master       3d20h  v1.25.0
      host-172-25-75-74  Ready   infra,worker 3d20h  v1.25.0
      host-172-25-75-79  Ready   worker       3d20h  v1.25.0
      host-172-25-75-86  Ready   worker       3d20h  v1.25.0
      host-172-25-75-98  Ready   infra,worker 3d20h  v1.25.0
      ```

      It can take several minutes for all nodes to report their state.

   b. If any nodes are in the **NotReady** state, log in to the nodes and remove all of the PEM files from the **/var/lib/kubelet/pki** directory on each node. You can SSH into the nodes or use the terminal window in the web console.

      ```
      $ ssh -i <ssh-key-path> core@<master-hostname>
      ```

      Sample pki directory

      ```
      sh-4.4# pwd
      /var/lib/kubelet/pki
      sh-4.4# ls
      kubelet-client-2022-04-28-11-24-09.pem  kubelet-server-2022-04-28-11-24-15.pem
      kubelet-client-current.pem              kubelet-server-current.pem
      ```

10. Restart the kubelet service on all control plane hosts.

    a. From the recovery host, run the following command:

       ```
       $ sudo systemctl restart kubelet.service
       ```

    b. Repeat this step on all other control plane hosts.

11. Approve the pending CSRs:

> **NOTE**
>
> Clusters with no worker nodes, such as single-node clusters or clusters consisting of three schedulable control plane nodes, will not have any pending CSRs to approve. You can skip all the commands listed in this step.

a. Get the list of current CSRs:

```
$ oc get csr
```

Example output

```
NAME        AGE    SIGNERNAME                         REQUESTOR
CONDITION
csr-2s94x   8m3s   kubernetes.io/kubelet-serving              system:node:<node_name>
Pending ❶
csr-4bd6t   8m3s   kubernetes.io/kubelet-serving              system:node:<node_name>
Pending ❷
csr-4hl85   13m    kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
❸
csr-zhhhp   3m8s   kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
❹
...
```

❶ ❶ ❷ A pending kubelet service CSR (for user-provisioned installations).

❸ ❹ A pending **node-bootstrapper** CSR.

b. Review the details of a CSR to verify that it is valid:

```
$ oc describe csr <csr_name> ❶
```

❶ **<csr_name>** is the name of a CSR from the list of current CSRs.

c. Approve each valid **node-bootstrapper** CSR:

```
$ oc adm certificate approve <csr_name>
```

d. For user-provisioned installations, approve each valid kubelet service CSR:

```
$ oc adm certificate approve <csr_name>
```

12. Verify that the single member control plane has started successfully.

a. From the recovery host, verify that the etcd container is running.

```
$ sudo crictl ps | grep etcd | egrep -v "operator|etcd-guard"
```

Example output

```
3ad41b7908e32
36f86e2eeaaffe662df0d21041eb22b8198e0e58abeeae8c743c3e6e977e8009
About a minute ago   Running          etcd                            0
7c05f8af362f0
```

b. From the recovery host, verify that the etcd pod is running.

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd
```

Example output

```
NAME                            READY  STATUS     RESTARTS  AGE
etcd-ip-10-0-143-125.ec2.internal      1/1    Running    1         2m47s
```

If the status is **Pending**, or the output lists more than one running etcd pod, wait a few minutes and check again.

13. If you are using the **OVNKubernetes** network plugin, delete the node objects that are associated with control plane hosts that are not the recovery control plane host.

```
$ oc delete node <non-recovery-controlplane-host-1> <non-recovery-controlplane-host-2>
```

14. Verify that the Cluster Network Operator (CNO) redeploys the OVN-Kubernetes control plane and that it no longer references the non-recovery controller IP addresses. To verify this result, regularly check the output of the following command. Wait until it returns an empty result before you proceed to restart the Open Virtual Network (OVN) Kubernetes pods on all of the hosts in the next step.

```
$ oc -n openshift-ovn-kubernetes get ds/ovnkube-master -o yaml | grep -E '<non-recovery_controller_ip_1>|<non-recovery_controller_ip_2>'
```

> **NOTE**
>
> It can take at least 5-10 minutes for the OVN-Kubernetes control plane to be redeployed and the previous command to return empty output.

15. If you are using the OVN-Kubernetes network plugin, restart the Open Virtual Network (OVN) Kubernetes pods on all of the hosts.

> **NOTE**
>
> Validating and mutating admission webhooks can reject pods. If you add any additional webhooks with the **failurePolicy** set to **Fail**, then they can reject pods and the restoration process can fail. You can avoid this by saving and deleting webhooks while restoring the cluster state. After the cluster state is restored successfully, you can enable the webhooks again.
>
> Alternatively, you can temporarily set the **failurePolicy** to **Ignore** while restoring the cluster state. After the cluster state is restored successfully, you can set the **failurePolicy** to **Fail**.

a. Remove the northbound database (nbdb) and southbound database (sbdb). Access the recovery host and the remaining control plane nodes by using Secure Shell (SSH) and run the following command:

```
$ sudo rm -f /var/lib/ovn/etc/*.db
```

b. Delete all OVN-Kubernetes control plane pods by running the following command:

```
$ oc delete pods -l app=ovnkube-master -n openshift-ovn-kubernetes
```

c. Ensure that any OVN-Kubernetes control plane pods are deployed again and are in a **Running** state by running the following command:

```
$ oc get pods -l app=ovnkube-master -n openshift-ovn-kubernetes
```

Example output

```
NAME                READY  STATUS   RESTARTS  AGE
ovnkube-master-nb24h  4/4    Running  0         48s
```

d. Delete all **ovnkube-node** pods by running the following command:

```
$ oc get pods -n openshift-ovn-kubernetes -o name | grep ovnkube-node | while read p ;
do oc delete $p -n openshift-ovn-kubernetes ; done
```

e. Ensure that all the **ovnkube-node** pods are deployed again and are in a **Running** state by running the following command:

```
$ oc get  pods -n openshift-ovn-kubernetes | grep ovnkube-node
```

16. Delete and re-create other non-recovery, control plane machines, one by one. After the machines are re-created, a new revision is forced and etcd automatically scales up.

- If you use a user-provisioned bare metal installation, you can re-create a control plane machine by using the same method that you used to originally create it. For more information, see "Installing a user-provisioned cluster on bare metal".

> **WARNING**
>
> Do not delete and re-create the machine for the recovery host.

- If you are running installer-provisioned infrastructure, or you used the Machine API to create your machines, follow these steps:

> **WARNING**
>
> Do not delete and re-create the machine for the recovery host.
>
> For bare metal installations on installer-provisioned infrastructure, control plane machines are not re-created. For more information, see "Replacing a bare-metal control plane node".

a. Obtain the machine for one of the lost control plane hosts.
   In a terminal that has access to the cluster as a cluster-admin user, run the following command:

   ```
   $ oc get machines -n openshift-machine-api -o wide
   ```

   Example output:

   ```
   NAME                              PHASE   TYPE      REGION    ZONE       AGE
   NODE                    PROVIDERID                      STATE
   clustername-8qw5l-master-0          Running  m4.xlarge us-east-1 us-east-1a
   3h37m  ip-10-0-131-183.ec2.internal  aws:///us-east-1a/i-0ec2782f8287dfb7e
   stopped 1
   clustername-8qw5l-master-1          Running  m4.xlarge us-east-1 us-east-1b
   3h37m  ip-10-0-143-125.ec2.internal  aws:///us-east-1b/i-096c349b700a19631
   running
   clustername-8qw5l-master-2          Running  m4.xlarge us-east-1 us-east-1c
   3h37m  ip-10-0-154-194.ec2.internal  aws:///us-east-1c/i-02626f1dba9ed5bba
   running
   clustername-8qw5l-worker-us-east-1a-wbtgd  Running m4.large   us-east-1  us-
   east-1a  3h28m  ip-10-0-129-226.ec2.internal  aws:///us-east-1a/i-
   010ef6279b4662ced   running
   clustername-8qw5l-worker-us-east-1b-lrdxb  Running m4.large   us-east-1  us-
   east-1b  3h28m  ip-10-0-144-248.ec2.internal  aws:///us-east-1b/i-
   0cb45ac45a166173b   running
   clustername-8qw5l-worker-us-east-1c-pkg26  Running m4.large   us-east-1  us-
   east-1c  3h28m  ip-10-0-170-181.ec2.internal  aws:///us-east-1c/i-
   06861c00007751b0a   running
   ```

   1 This is the control plane machine for the lost control plane host, **ip-10-0-131-183.ec2.internal**.

b. Save the machine configuration to a file on your file system:

   ```
   $ oc get machine clustername-8qw5l-master-0 \    1
       -n openshift-machine-api \
       -o yaml \
       > new-master-machine.yaml
   ```

   1 Specify the name of the control plane machine for the lost control plane host.

c. Edit the **new-master-machine.yaml** file that was created in the previous step to assign a new name and remove unnecessary fields.

i. Remove the entire **status** section:

```
status:
  addresses:
  - address: 10.0.131.183
    type: InternalIP
  - address: ip-10-0-131-183.ec2.internal
    type: InternalDNS
  - address: ip-10-0-131-183.ec2.internal
    type: Hostname
  lastUpdated: "2020-04-20T17:44:29Z"
  nodeRef:
    kind: Node
    name: ip-10-0-131-183.ec2.internal
    uid: acca4411-af0d-4387-b73e-52b2484295ad
  phase: Running
  providerStatus:
    apiVersion: awsproviderconfig.openshift.io/v1beta1
    conditions:
    - lastProbeTime: "2020-04-20T16:53:50Z"
      lastTransitionTime: "2020-04-20T16:53:50Z"
      message: machine successfully created
      reason: MachineCreationSucceeded
      status: "True"
      type: MachineCreation
    instanceId: i-0fdb85790d76d0c3f
    instanceState: stopped
    kind: AWSMachineProviderStatus
```

ii. Change the **metadata.name** field to a new name.
It is recommended to keep the same base name as the old machine and change the ending number to the next available number. In this example, **clustername-8qw5l-master-0** is changed to **clustername-8qw5l-master-3**:

```
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  ...
  name: clustername-8qw5l-master-3
  ...
```

iii. Remove the **spec.providerID** field:

```
providerID: aws:///us-east-1a/i-0fdb85790d76d0c3f
```

iv. Remove the **metadata.annotations** and **metadata.generation** fields:

```
annotations:
  machine.openshift.io/instance-state: running
...
generation: 2
```

v. Remove the **metadata.resourceVersion** and **metadata.uid** fields:

```
resourceVersion: "13291"
uid: a282eb70-40a2-4e89-8009-d05dd420d31a
```

d. Delete the machine of the lost control plane host:

```
$ oc delete machine -n openshift-machine-api clustername-8qw5l-master-0 1
```

1 Specify the name of the control plane machine for the lost control plane host.

e. Verify that the machine was deleted:

```
$ oc get machines -n openshift-machine-api -o wide
```

Example output:

```
NAME                        PHASE    TYPE      REGION    ZONE      AGE
NODE              PROVIDERID                     STATE
clustername-8qw5l-master-1             Running  m4.xlarge  us-east-1  us-east-1b
3h37m  ip-10-0-143-125.ec2.internal  aws:///us-east-1b/i-096c349b700a19631
running
clustername-8qw5l-master-2             Running  m4.xlarge  us-east-1  us-east-1c
3h37m  ip-10-0-154-194.ec2.internal  aws:///us-east-1c/i-02626f1dba9ed5bba
running
clustername-8qw5l-worker-us-east-1a-wbtgd  Running  m4.large   us-east-1  us-
east-1a  3h28m  ip-10-0-129-226.ec2.internal  aws:///us-east-1a/i-
010ef6279b4662ced   running
clustername-8qw5l-worker-us-east-1b-lrdxb  Running  m4.large   us-east-1  us-
east-1b  3h28m  ip-10-0-144-248.ec2.internal  aws:///us-east-1b/i-
0cb45ac45a166173b   running
clustername-8qw5l-worker-us-east-1c-pkg26  Running  m4.large   us-east-1  us-
east-1c  3h28m  ip-10-0-170-181.ec2.internal  aws:///us-east-1c/i-
06861c00007751b0a   running
```

f. Create a machine by using the **new-master-machine.yaml** file:

```
$ oc apply -f new-master-machine.yaml
```

g. Verify that the new machine has been created:

```
$ oc get machines -n openshift-machine-api -o wide
```

Example output:

```
NAME                        PHASE    TYPE      REGION    ZONE
AGE    NODE               PROVIDERID                     STATE
clustername-8qw5l-master-1             Running    m4.xlarge  us-east-1  us-east-
1b  3h37m  ip-10-0-143-125.ec2.internal  aws:///us-east-1b/i-096c349b700a19631
running
clustername-8qw5l-master-2             Running    m4.xlarge  us-east-1  us-east-
1c  3h37m  ip-10-0-154-194.ec2.internal  aws:///us-east-1c/i-02626f1dba9ed5bba
running
```

```
clustername-8qw5l-master-3              Provisioning  m4.xlarge  us-east-1  us-east-
1a  85s    ip-10-0-173-171.ec2.internal    aws:///us-east-1a/i-015b0888fe17bc2c8
running
```

```
clustername-8qw5l-worker-us-east-1a-wbtgd  Running      m4.large   us-east-1
us-east-1a  3h28m  ip-10-0-129-226.ec2.internal  aws:///us-east-1a/i-
010ef6279b4662ced   running
clustername-8qw5l-worker-us-east-1b-lrdxb  Running      m4.large   us-east-1  us-
east-1b  3h28m  ip-10-0-144-248.ec2.internal  aws:///us-east-1b/i-
0cb45ac45a166173b   running
clustername-8qw5l-worker-us-east-1c-pkg26  Running      m4.large   us-east-1
us-east-1c  3h28m  ip-10-0-170-181.ec2.internal  aws:///us-east-1c/i-
06861c00007751b0a   running
```

 The new machine, **clustername-8qw5l-master-3** is being created and is ready after the phase changes from **Provisioning** to **Running**.

It might take a few minutes for the new machine to be created. The etcd cluster Operator will automatically sync when the machine or node returns to a healthy state.

   h. Repeat these steps for each lost control plane host that is not the recovery host.

17. Turn off the quorum guard by entering the following command:

    ```
    $ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides":
    {"useUnsupportedUnsafeNonHANonProductionUnstableEtcd": true}}}'
    ```

    This command ensures that you can successfully re-create secrets and roll out the static pods.

18. In a separate terminal window within the recovery host, export the recovery **kubeconfig** file by running the following command:

    ```
    $ export KUBECONFIG=/etc/kubernetes/static-pod-resources/kube-apiserver-
    certs/secrets/node-kubeconfigs/localhost-recovery.kubeconfig
    ```

19. Force etcd redeployment.
    In the same terminal window where you exported the recovery **kubeconfig** file, run the following command:

    ```
    $ oc patch etcd cluster -p='{"spec": {"forceRedeploymentReason": "recovery-'"$( date --rfc-
    3339=ns )"'"}}' --type=merge
    ```


     The **forceRedeploymentReason** value must be unique, which is why a timestamp is appended.

    When the etcd cluster Operator performs a redeployment, the existing nodes are started with new pods similar to the initial bootstrap scale up.

20. Turn the quorum guard back on by entering the following command:

    ```
    $ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides": null}}'
    ```

21. You can verify that the **unsupportedConfigOverrides** section is removed from the object by entering this command:

```
$ oc get etcd/cluster -oyaml
```

22. Verify all nodes are updated to the latest revision.
    In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc get etcd -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

Review the **NodeInstallerProgressing** status condition for etcd to verify that all nodes are at the latest revision. The output shows **AllNodesAtLatestRevision** upon successful update:

```
AllNodesAtLatestRevision
3 nodes are at revision 7 1
```

1   In this example, the latest revision number is **7**.

If the output includes multiple revision numbers, such as **2 nodes are at revision 6; 1 nodes are at revision 7**, this means that the update is still in progress. Wait a few minutes and try again.

23. After etcd is redeployed, force new rollouts for the control plane. The Kubernetes API server will reinstall itself on the other nodes because the kubelet is connected to API servers using an internal load balancer.
    In a terminal that has access to the cluster as a **cluster-admin** user, run the following commands.

    a. Force a new rollout for the Kubernetes API server:

```
$ oc patch kubeapiserver cluster -p='{"spec": {"forceRedeploymentReason": "recovery-
"$( date --rfc-3339=ns )'"}}' --type=merge
```

    Verify all nodes are updated to the latest revision.

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

    Review the **NodeInstallerProgressing** status condition to verify that all nodes are at the latest revision. The output shows **AllNodesAtLatestRevision** upon successful update:

```
AllNodesAtLatestRevision
3 nodes are at revision 7 1
```

1   In this example, the latest revision number is **7**.

    If the output includes multiple revision numbers, such as **2 nodes are at revision 6; 1 nodes are at revision 7**, this means that the update is still in progress. Wait a few minutes and try again.

    b. Force a new rollout for the Kubernetes controller manager:

```
$ oc patch kubecontrollermanager cluster -p='{"spec": {"forceRedeploymentReason":
"recovery-"$( date --rfc-3339=ns )'"}}' --type=merge
```

Verify all nodes are updated to the latest revision.

```
$ oc get kubecontrollermanager -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

Review the **NodeInstallerProgressing** status condition to verify that all nodes are at the latest revision. The output shows **AllNodesAtLatestRevision** upon successful update:

```
AllNodesAtLatestRevision
3 nodes are at revision 7 ❶
```

❶ In this example, the latest revision number is **7**.

If the output includes multiple revision numbers, such as **2 nodes are at revision 6; 1 nodes are at revision 7**, this means that the update is still in progress. Wait a few minutes and try again.

c. Force a new rollout for the Kubernetes scheduler:

```
$ oc patch kubescheduler cluster -p='{"spec": {"forceRedeploymentReason": "recovery-
"'"$( date --rfc-3339=ns )"'"}}' --type=merge
```

Verify all nodes are updated to the latest revision.

```
$ oc get kubescheduler -o=jsonpath='{range .items[0].status.conditions[?
(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

Review the **NodeInstallerProgressing** status condition to verify that all nodes are at the latest revision. The output shows **AllNodesAtLatestRevision** upon successful update:

```
AllNodesAtLatestRevision
3 nodes are at revision 7 ❶
```

❶ In this example, the latest revision number is **7**.

If the output includes multiple revision numbers, such as **2 nodes are at revision 6; 1 nodes are at revision 7**, this means that the update is still in progress. Wait a few minutes and try again.

24. Verify that all control plane hosts have started and joined the cluster.
In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd
```

Example output

```
etcd-ip-10-0-143-125.ec2.internal          2/2    Running    0        9h
etcd-ip-10-0-154-194.ec2.internal          2/2    Running    0        9h
etcd-ip-10-0-173-171.ec2.internal          2/2    Running    0        9h
```

To ensure that all workloads return to normal operation following a recovery procedure, restart each pod that stores Kubernetes API information. This includes OpenShift Container Platform components such as routers, Operators, and third-party components.

> **NOTE**
>
> On completion of the previous procedural steps, you might need to wait a few minutes for all services to return to their restored state. For example, authentication by using **oc login** might not immediately work until the OAuth server pods are restarted.
>
> Consider using the **system:admin kubeconfig** file for immediate authentication. This method basis its authentication on SSL/TLS client certificates as against OAuth tokens. You can authenticate with this file by issuing the following command:
>
> ```
> $ export KUBECONFIG=<installation_directory>/auth/kubeconfig
> ```
>
> Issue the following command to display your authenticated user name:
>
> ```
> $ oc whoami
> ```

Additional resources

- Installing a user-provisioned cluster on bare metal

- Replacing a bare-metal control plane node

6.14.7. Issues and workarounds for restoring a persistent storage state

If your OpenShift Container Platform cluster uses persistent storage of any form, a state of the cluster is typically stored outside etcd. It might be an Elasticsearch cluster running in a pod or a database running in a **StatefulSet** object. When you restore from an etcd backup, the status of the workloads in OpenShift Container Platform is also restored. However, if the etcd snapshot is old, the status might be invalid or outdated.
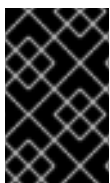
> **IMPORTANT**
>
> The contents of persistent volumes (PVs) are never part of the etcd snapshot. When you restore an OpenShift Container Platform cluster from an etcd snapshot, non-critical workloads might gain access to critical data, or vice-versa.

The following are some example scenarios that produce an out-of-date status:

- MySQL database is running in a pod backed up by a PV object. Restoring OpenShift Container Platform from an etcd snapshot does not bring back the volume on the storage provider, and does not produce a running MySQL pod, despite the pod repeatedly attempting to start. You must manually restore this pod by restoring the volume on the storage provider, and then editing the PV to point to the new volume.

- Pod P1 is using volume A, which is attached to node X. If the etcd snapshot is taken while another pod uses the same volume on node Y, then when the etcd restore is performed, pod P1 might not be able to start correctly due to the volume still being attached to node Y. OpenShift Container Platform is not aware of the attachment, and does not automatically detach it. When this occurs, the volume must be manually detached from node Y so that the volume can attach on node X, and then pod P1 can start.

- Cloud provider or storage provider credentials were updated after the etcd snapshot was taken. This causes any CSI drivers or Operators that depend on the those credentials to not work. You might have to manually update the credentials required by those drivers or Operators.

- A device is removed or renamed from OpenShift Container Platform nodes after the etcd snapshot is taken. The Local Storage Operator creates symlinks for each PV that it manages from **/dev/disk/by-id** or **/dev** directories. This situation might cause the local PVs to refer to devices that no longer exist.
  To fix this problem, an administrator must:

  1. Manually remove the PVs with invalid devices.

  2. Remove symlinks from respective nodes.

  3. Delete **LocalVolume** or **LocalVolumeSet** objects (see *Storage → Configuring persistent storage → Persistent storage using local volumes → Deleting the Local Storage Operator Resources*).

6.15. POD DISRUPTION BUDGETS

Understand and configure pod disruption budgets.

6.15.1. Understanding how to use pod disruption budgets to specify the number of pods that must be up

A *pod disruption budget* allows the specification of safety constraints on pods during operations, such as draining a node for maintenance.

PodDisruptionBudget is an API object that specifies the minimum number or percentage of replicas that must be up at a time. Setting these in projects can be helpful during node maintenance (such as scaling a cluster down or a cluster upgrade) and is only honored on voluntary evictions (not on node failures).

A **PodDisruptionBudget** object's configuration consists of the following key parts:

- A label selector, which is a label query over a set of pods.

- An availability level, which specifies the minimum number of pods that must be available simultaneously, either:

  - **minAvailable** is the number of pods must always be available, even during a disruption.

  - **maxUnavailable** is the number of pods can be unavailable during a disruption.

> **NOTE**
>
> **Available** refers to the number of pods that has condition **Ready=True**. **Ready=True** refers to the pod that is able to serve requests and should be added to the load balancing pools of all matching services.
>
> A **maxUnavailable** of **0%** or **0** or a **minAvailable** of **100%** or equal to the number of replicas is permitted but can block nodes from being drained.

You can check for pod disruption budgets across all projects with the following:

```
$ oc get poddisruptionbudget --all-namespaces
```

Example output

```
NAMESPACE                      NAME                        MIN AVAILABLE   MAX UNAVAILABLE
ALLOWED DISRUPTIONS   AGE
openshift-apiserver            openshift-apiserver-pdb        N/A        1        1
121m
openshift-cloud-controller-manager    aws-cloud-controller-manager    1        N/A        1
125m
openshift-cloud-credential-operator    pod-identity-webhook        1        N/A        1
117m
openshift-cluster-csi-drivers        aws-ebs-csi-driver-controller-pdb    N/A        1        1
121m
openshift-cluster-storage-operator    csi-snapshot-controller-pdb        N/A        1        1
122m
openshift-cluster-storage-operator    csi-snapshot-webhook-pdb        N/A        1        1
122m
openshift-console              console                     N/A        1        1
116m
#...
```

The **PodDisruptionBudget** is considered healthy when there are at least **minAvailable** pods running in the system. Every pod above that limit can be evicted.

> **NOTE**
>
> Depending on your pod priority and preemption settings, lower-priority pods might be removed despite their pod disruption budget requirements.

6.15.2. Specifying the number of pods that must be up with pod disruption budgets

You can use a **PodDisruptionBudget** object to specify the minimum number or percentage of replicas that must be up at a time.

Procedure

To configure a pod disruption budget:

1. Create a YAML file with the an object definition similar to the following:

   ```
   apiVersion: policy/v1    1
   kind: PodDisruptionBudget
   metadata:
     name: my-pdb
   spec:
     minAvailable: 2    2
     selector:    3
       matchLabels:
         name: my-pod
   ```

   1 **PodDisruptionBudget** is part of the **policy/v1** API group.

   2 The minimum number of pods that must be available simultaneously. This can be either an integer or a string specifying a percentage, for example, **20%**.

integer or a string specifying a percentage, for example, **20%**.

3 A label query over a set of resources. The result of **matchLabels** and **matchExpressions** are logically conjoined. Leave this paramter blank, for example **selector {}**, to select all pods in the project.

Or:

```
apiVersion: policy/v1 1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  maxUnavailable: 25% 2
  selector: 3
    matchLabels:
      name: my-pod
```

1 **PodDisruptionBudget** is part of the **policy/v1** API group.

2 The maximum number of pods that can be unavailable simultaneously. This can be either an integer or a string specifying a percentage, for example, **20%**.

3 A label query over a set of resources. The result of **matchLabels** and **matchExpressions** are logically conjoined. Leave this paramter blank, for example **selector {}**, to select all pods in the project.

2. Run the following command to add the object to project:

```
$ oc create -f </path/to/file> -n <project_name>
```

6.16. ROTATING OR REMOVING CLOUD PROVIDER CREDENTIALS

After installing OpenShift Container Platform, some organizations require the rotation or removal of the cloud provider credentials that were used during the initial installation.

To allow the cluster to use the new credentials, you must update the secrets that the Cloud Credential Operator (CCO) uses to manage cloud provider credentials.

6.16.1. Rotating cloud provider credentials with the Cloud Credential Operator utility

The Cloud Credential Operator (CCO) utility **ccoctl** supports updating secrets for clusters installed on IBM Cloud.

6.16.1.1. Rotating API keys for IBM Cloud

You can rotate API keys for your existing service IDs and update the corresponding secrets.

Prerequisites

- You have configured the **ccoctl** binary.

- You have existing service IDs in a live OpenShift Container Platform cluster installed on IBM Cloud.

Procedure

- Use the **ccoctl** utility to rotate your API keys for the service IDs and update the secrets:

```
$ ccoctl ibmcloud refresh-keys \
    --kubeconfig <openshift_kubeconfig_file> \ 1
    --credentials-requests-dir <path_to_credential_requests_directory> \ 2
    --name <name> 3
```

1. The **kubeconfig** file associated with the cluster. For example, **<installation_directory>/auth/kubeconfig**.

2. The directory where the credential requests are stored.

3. The name of the OpenShift Container Platform cluster.
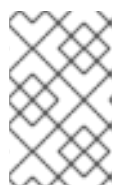
> **NOTE**
>
> If your cluster uses Technology Preview features that are enabled by the **TechPreviewNoUpgrade** feature set, you must include the **--enable-tech-preview** parameter.

6.16.2. Rotating cloud provider credentials manually

If your cloud provider credentials are changed for any reason, you must manually update the secret that the Cloud Credential Operator (CCO) uses to manage cloud provider credentials.

The process for rotating cloud credentials depends on the mode that the CCO is configured to use. After you rotate credentials for a cluster that is using mint mode, you must manually remove the component credentials that were created by the removed credential.

Prerequisites

- Your cluster is installed on a platform that supports rotating cloud credentials manually with the CCO mode that you are using:

  - For mint mode, Amazon Web Services (AWS) and Google Cloud Platform (GCP) are supported.

  - For passthrough mode, Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), Red Hat OpenStack Platform (RHOSP), Red Hat Virtualization (RHV), and VMware vSphere are supported.

- You have changed the credentials that are used to interface with your cloud provider.

- The new credentials have sufficient permissions for the mode CCO is configured to use in your cluster.

Procedure

1. In the **Administrator** perspective of the web console, navigate to **Workloads → Secrets**.

2. In the table on the **Secrets** page, find the root secret for your cloud provider.

Platform	Secret name
AWS	**aws-creds**
Azure	**azure-credentials**
GCP	**gcp-credentials**
RHOSP	**openstack-credentials**
RHV	**ovirt-credentials**
VMware vSphere	**vsphere-creds**

3. Click the **Options** menu ⋮ in the same row as the secret and select **Edit Secret**.

4. Record the contents of the **Value** field or fields. You can use this information to verify that the value is different after updating the credentials.

5. Update the text in the **Value** field or fields with the new authentication information for your cloud provider, and then click **Save**.

6. If you are updating the credentials for a vSphere cluster that does not have the vSphere CSI Driver Operator enabled, you must force a rollout of the Kubernetes controller manager to apply the updated credentials.
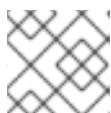
   > **NOTE**
   >
   > If the vSphere CSI Driver Operator is enabled, this step is not required.

   To apply the updated vSphere credentials, log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role and run the following command:

   ```
   $ oc patch kubecontrollermanager cluster \
     -p='{"spec": {"forceRedeploymentReason": "recovery-"$( date )"""}}' \
     --type=merge
   ```

   While the credentials are rolling out, the status of the Kubernetes Controller Manager Operator reports **Progressing=true**. To view the status, run the following command:

   ```
   $ oc get co kube-controller-manager
   ```

7. If the CCO for your cluster is configured to use mint mode, delete each component secret that is referenced by the individual **CredentialsRequest** objects.

   a. Log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role.

   b. Get the names and namespaces of all referenced component secrets:

```
$ oc -n openshift-cloud-credential-operator get CredentialsRequest \
  -o json | jq -r '.items[] | select (.spec.providerSpec.kind=="<provider_spec>") |
.spec.secretRef'
```

where **<provider_spec>** is the corresponding value for your cloud provider:

- AWS: **AWSProviderSpec**

- GCP: **GCPProviderSpec**

Partial example output for AWS

```
{
  "name": "ebs-cloud-credentials",
  "namespace": "openshift-cluster-csi-drivers"
}
{
  "name": "cloud-credential-operator-iam-ro-creds",
  "namespace": "openshift-cloud-credential-operator"
}
```

c. Delete each of the referenced component secrets:

```
$ oc delete secret <secret_name> \      ❶
  -n <secret_namespace>      ❷
```

❶ Specify the name of a secret.

❷ Specify the namespace that contains the secret.

Example deletion of an AWS secret

```
$ oc delete secret ebs-cloud-credentials -n openshift-cluster-csi-drivers
```

You do not need to manually delete the credentials from your provider console. Deleting the referenced component secrets will cause the CCO to delete the existing credentials from the platform and create new ones.

Verification

To verify that the credentials have changed:

1. In the **Administrator** perspective of the web console, navigate to **Workloads → Secrets**.

2. Verify that the contents of the **Value** field or fields have changed.

Additional resources

- vSphere CSI Driver Operator

6.16.3. Removing cloud provider credentials

After installing an OpenShift Container Platform cluster with the Cloud Credential Operator (CCO) in

mint mode, you can remove the administrator-level credential secret from the **kube-system** namespace in the cluster. The administrator-level credential is required only during changes that require its elevated permissions, such as upgrades.

> **NOTE**
>
> Prior to a non z-stream upgrade, you must reinstate the credential secret with the administrator-level credential. If the credential is not present, the upgrade might be blocked.

Prerequisites

- Your cluster is installed on a platform that supports removing cloud credentials from the CCO. Supported platforms are AWS and GCP.

Procedure

1. In the **Administrator** perspective of the web console, navigate to **Workloads → Secrets**.

2. In the table on the **Secrets** page, find the root secret for your cloud provider.

Platform	Secret name
AWS	**aws-creds**
GCP	**gcp-credentials**

3. Click the **Options** menu ⋮ in the same row as the secret and select **Delete Secret**.

Additional resources

- [About the Cloud Credential Operator](#)

- [Amazon Web Services (AWS) secret format](#)

- [Microsoft Azure secret format](#)

- [Google Cloud Platform (GCP) secret format](#)

6.17. CONFIGURING IMAGE STREAMS FOR A DISCONNECTED CLUSTER

After installing OpenShift Container Platform in a disconnected environment, configure the image streams for the Cluster Samples Operator and the **must-gather** image stream.

6.17.1. Cluster Samples Operator assistance for mirroring

During installation, OpenShift Container Platform creates a config map named **imagestreamtag-to-image** in the **openshift-cluster-samples-operator** namespace. The **imagestreamtag-to-image** config map contains an entry, the populating image, for each image stream tag.

The format of the key for each entry in the data field in the config map is
<image_stream_name>_<image_stream_tag_name>.

During a disconnected installation of OpenShift Container Platform, the status of the Cluster Samples Operator is set to **Removed**. If you choose to change it to **Managed**, it installs samples.

> **NOTE**
>
> The use of samples in a network-restricted or discontinued environment may require access to services external to your network. Some example services include: Github, Maven Central, npm, RubyGems, PyPi and others. There might be additional steps to take that allow the cluster samples operators's objects to reach the services they require.

You can use this config map as a reference for which images need to be mirrored for your image streams to import.

- While the Cluster Samples Operator is set to **Removed**, you can create your mirrored registry, or determine which existing mirrored registry you want to use.

- Mirror the samples you want to the mirrored registry using the new config map as your guide.

- Add any of the image streams you did not mirror to the **skippedImagestreams** list of the Cluster Samples Operator configuration object.

- Set **samplesRegistry** of the Cluster Samples Operator configuration object to the mirrored registry.

- Then set the Cluster Samples Operator to **Managed** to install the image streams you have mirrored.

6.17.2. Using Cluster Samples Operator image streams with alternate or mirrored registries

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

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

> **IMPORTANT**
>
> The Cluster Samples Operator must be set to **Managed** in a disconnected environment. To install the image streams, you have a mirrored registry.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

- Create a pull secret for your mirror registry.

Procedure

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

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

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

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

3. Create the cluster's image configuration object:

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

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

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

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

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

   > **NOTE**
   >
   > This is required because the image stream import process does not use the mirror or search mechanism at this time.

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

   > **NOTE**
   >
   > The Cluster Samples Operator issues alerts if image stream imports are failing but the Cluster Samples Operator is either periodically retrying or does not appear to be retrying them.

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

6.17.3. Preparing your cluster to gather support data

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

Procedure

1. If you have not added your mirror registry's trusted CA to your cluster's image configuration object as part of the Cluster Samples Operator configuration, perform the following steps:

   a. Create the cluster's image configuration object:

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

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

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

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

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

When running the **oc adm must-gather** command, use the **--image** flag and point to the payload image, as in the following example:

```
$ oc adm must-gather --image=$(oc adm release info --image-for must-gather)
```

6.18. CONFIGURING PERIODIC IMPORTING OF CLUSTER SAMPLE OPERATOR IMAGE STREAM TAGS

You can ensure that you always have access to the latest versions of the Cluster Sample Operator images by periodically importing the image stream tags when new versions become available.

Procedure

1. Fetch all the imagestreams in the **openshift** namespace by running the following command:

   ```
   oc get imagestreams -nopenshift
   ```

2. Fetch the tags for every imagestream in the **openshift** namespace by running the following command:

   ```
   $ oc get is <image-stream-name> -o jsonpath="{range .spec.tags[*]}{.name}{'\t'}{.from.name}
   {'\n'}{end}" -nopenshift
   ```

   For example:

   ```
   $ oc get is ubi8-openjdk-17 -o jsonpath="{range .spec.tags[*]}{.name}{'\t'}{.from.name}{'\n'}
   {end}" -nopenshift
   ```

   Example output

```
1.11 registry.access.redhat.com/ubi8/openjdk-17:1.11
1.12 registry.access.redhat.com/ubi8/openjdk-17:1.12
```

3. Schedule periodic importing of images for each tag present in the image stream by running the following command:

```
$ oc tag <repository/image> <image-stream-name:tag> --scheduled -nopenshift
```

For example:

```
$ oc tag registry.access.redhat.com/ubi8/openjdk-17:1.11 ubi8-openjdk-17:1.11 --scheduled -nopenshift
$ oc tag registry.access.redhat.com/ubi8/openjdk-17:1.12 ubi8-openjdk-17:1.12 --scheduled -nopenshift
```

This command causes OpenShift Container Platform to periodically update this particular image stream tag. This period is a cluster-wide setting set to 15 minutes by default.

4. Verify the scheduling status of the periodic import by running the following command:

```
oc get imagestream <image-stream-name> -o jsonpath="{range .spec.tags[*]}Tag: {.name}{'\t'}Scheduled: {.importPolicy.scheduled}{'\n'}{end}" -nopenshift
```

For example:

```
oc get imagestream ubi8-openjdk-17 -o jsonpath="{range .spec.tags[*]}Tag: {.name}{'\t'}Scheduled: {.importPolicy.scheduled}{'\n'}{end}" -nopenshift
```

Example output

```
Tag: 1.11 Scheduled: true
Tag: 1.12 Scheduled: true
```

CHAPTER 7. POSTINSTALLATION NODE TASKS

After installing OpenShift Container Platform, you can further expand and customize your cluster to your requirements through certain node tasks.

7.1. ADDING RHEL COMPUTE MACHINES TO AN OPENSHIFT CONTAINER PLATFORM CLUSTER

Understand and work with RHEL compute nodes.

7.1.1. About adding RHEL compute nodes to a cluster

In OpenShift Container Platform 4.12, you have the option of using Red Hat Enterprise Linux (RHEL) machines as compute machines in your cluster if you use a user-provisioned or installer-provisioned infrastructure installation on the **x86_64** architecture. You must use Red Hat Enterprise Linux CoreOS (RHCOS) machines for the control plane machines in your cluster.

If you choose to use RHEL compute machines in your cluster, you are responsible for all operating system life cycle management and maintenance. You must perform system updates, apply patches, and complete all other required tasks.

For installer-provisioned infrastructure clusters, you must manually add RHEL compute machines because automatic scaling in installer-provisioned infrastructure clusters adds Red Hat Enterprise Linux CoreOS (RHCOS) compute machines by default.

> **IMPORTANT**
>
> - Because removing OpenShift Container Platform from a machine in the cluster requires destroying the operating system, you must use dedicated hardware for any RHEL machines that you add to the cluster.
>
> - Swap memory is disabled on all RHEL machines that you add to your OpenShift Container Platform cluster. You cannot enable swap memory on these machines.

You must add any RHEL compute machines to the cluster after you initialize the control plane.

7.1.2. System requirements for RHEL compute nodes

The Red Hat Enterprise Linux (RHEL) compute machine hosts in your OpenShift Container Platform environment must meet the following minimum hardware specifications and system-level requirements:

- You must have an active OpenShift Container Platform subscription on your Red Hat account. If you do not, contact your sales representative for more information.

- Production environments must provide compute machines to support your expected workloads. As a cluster administrator, you must calculate the expected workload and add about 10% for overhead. For production environments, allocate enough resources so that a node host failure does not affect your maximum capacity.

- Each system must meet the following hardware requirements:

  - Physical or virtual system, or an instance running on a public or private IaaS.

  - Base OS: RHEL 8.6 and later with "Minimal" installation option.

> **IMPORTANT**
>
> Adding RHEL 7 compute machines to an OpenShift Container Platform cluster is not supported.
>
> If you have RHEL 7 compute machines that were previously supported in a past OpenShift Container Platform version, you cannot upgrade them to RHEL 8. You must deploy new RHEL 8 hosts, and the old RHEL 7 hosts should be removed. See the "Deleting nodes" section for more information.
>
> 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.

- If you deployed OpenShift Container Platform in FIPS mode, you must enable FIPS on the RHEL machine before you boot it. See Installing a RHEL 8 system with FIPS mode enabled in the RHEL 8 documentation.

> **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. The use of FIPS validated or Modules In Process cryptographic libraries is only supported on OpenShift Container Platform deployments on the **x86_64**, **ppc64le**, and **s390x** architectures.

- NetworkManager 1.0 or later.

- 1 vCPU.

- Minimum 8 GB RAM.

- Minimum 15 GB hard disk space for the file system containing **/var**/.

- Minimum 1 GB hard disk space for the file system containing **/usr**/**local**/**bin**/.

- Minimum 1 GB hard disk space for the file system containing its temporary directory. The temporary system directory is determined according to the rules defined in the tempfile module in the Python standard library.

  - Each system must meet any additional requirements for your system provider. For example, if you installed your cluster on VMware vSphere, your disks must be configured according to its storage guidelines and the **disk.enableUUID=true** attribute must be set.

  - Each system must be able to access the cluster's API endpoints by using DNS-resolvable hostnames. Any network security access control that is in place must allow system access to the cluster's API service endpoints.

Additional resources

- Deleting nodes

7.1.2.1. Certificate signing requests management

Because your cluster has limited access to automatic machine management when you use infrastructure that you provision, you must provide a mechanism for approving cluster certificate signing requests (CSRs) after installation. The **kube-controller-manager** only approves the kubelet client CSRs. The **machine-approver** cannot guarantee the validity of a serving certificate that is requested by using kubelet credentials because it cannot confirm that the correct machine issued the request. You must determine and implement a method of verifying the validity of the kubelet serving certificate requests and approving them.

7.1.3. Preparing the machine to run the playbook

Before you can add compute machines that use Red Hat Enterprise Linux (RHEL) as the operating system to an OpenShift Container Platform 4.12 cluster, you must prepare a RHEL 8 machine to run an Ansible playbook that adds the new node to the cluster. This machine is not part of the cluster but must be able to access it.

Prerequisites

- Install the OpenShift CLI (**oc**) on the machine that you run the playbook on.

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

Procedure

1. Ensure that the **kubeconfig** file for the cluster and the installation program that you used to install the cluster are on the RHEL 8 machine. One way to accomplish this is to use the same machine that you used to install the cluster.

2. Configure the machine to access all of the RHEL hosts that you plan to use as compute machines. You can use any method that your company allows, including a bastion with an SSH proxy or a VPN.

3. Configure a user on the machine that you run the playbook on that has SSH access to all of the RHEL hosts.
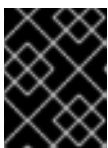
   > **IMPORTANT**
   >
   > If you use SSH key-based authentication, you must manage the key with an SSH agent.

4. If you have not already done so, register the machine with RHSM and attach a pool with an **OpenShift** subscription to it:

   a. Register the machine with RHSM:

   ```
   # subscription-manager register --username=<user_name> --password=<password>
   ```

   b. Pull the latest subscription data from RHSM:

   ```
   # subscription-manager refresh
   ```

   c. List the available subscriptions:

   ```
   # subscription-manager list --available --matches '*OpenShift*'
   ```

d. In the output for the previous command, find the pool ID for an OpenShift Container Platform subscription and attach it:

```
# subscription-manager attach --pool=<pool_id>
```

5. Enable the repositories required by OpenShift Container Platform 4.12:

```
# subscription-manager repos \
    --enable="rhel-8-for-x86_64-baseos-rpms" \
    --enable="rhel-8-for-x86_64-appstream-rpms" \
    --enable="rhocp-4.12-for-rhel-8-x86_64-rpms"
```

6. Install the required packages, including **openshift-ansible**:

```
# yum install openshift-ansible openshift-clients jq
```

The **openshift-ansible** package provides installation program utilities and pulls in other packages that you require to add a RHEL compute node to your cluster, such as Ansible, playbooks, and related configuration files. The **openshift-clients** provides the **oc** CLI, and the **jq** package improves the display of JSON output on your command line.

7.1.4. Preparing a RHEL compute node

Before you add a Red Hat Enterprise Linux (RHEL) machine to your OpenShift Container Platform cluster, you must register each host with Red Hat Subscription Manager (RHSM), attach an active OpenShift Container Platform subscription, and enable the required repositories. Ensure **NetworkManager** is enabled and configured to control all interfaces on the host.

1. On each host, register with RHSM:

```
# subscription-manager register --username=<user_name> --password=<password>
```

2. Pull the latest subscription data from RHSM:

```
# subscription-manager refresh
```

3. List the available subscriptions:

```
# subscription-manager list --available --matches '*OpenShift*'
```

4. In the output for the previous command, find the pool ID for an OpenShift Container Platform subscription and attach it:

```
# subscription-manager attach --pool=<pool_id>
```

5. Disable all yum repositories:

a. Disable all the enabled RHSM repositories:

```
# subscription-manager repos --disable="*"
```

b. List the remaining yum repositories and note their names under **repo id**, if any:

```
# yum repolist
```

c. Use **yum-config-manager** to disable the remaining yum repositories:

```
# yum-config-manager --disable <repo_id>
```

Alternatively, disable all repositories:

```
# yum-config-manager --disable \*
```

Note that this might take a few minutes if you have a large number of available repositories

6. Enable only the repositories required by OpenShift Container Platform 4.12:

```
# subscription-manager repos \
    --enable="rhel-8-for-x86_64-baseos-rpms" \
    --enable="rhel-8-for-x86_64-appstream-rpms" \
    --enable="rhocp-4.12-for-rhel-8-x86_64-rpms" \
    --enable="fast-datapath-for-rhel-8-x86_64-rpms"
```

7. Stop and disable firewalld on the host:

```
# systemctl disable --now firewalld.service
```
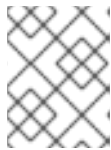
> **NOTE**
>
> You must not enable firewalld later. If you do, you cannot access OpenShift Container Platform logs on the worker.

7.1.5. Adding a RHEL compute machine to your cluster

You can add compute machines that use Red Hat Enterprise Linux as the operating system to an OpenShift Container Platform 4.12 cluster.

Prerequisites

- You installed the required packages and performed the necessary configuration on the machine that you run the playbook on.

- You prepared the RHEL hosts for installation.

Procedure

Perform the following steps on the machine that you prepared to run the playbook:

1. Create an Ansible inventory file that is named /**<path>**/**inventory**/**hosts** that defines your compute machine hosts and required variables:

```
[all:vars]
ansible_user=root 1
#ansible_become=True 2

openshift_kubeconfig_path="~/.kube/config" 3
```

```
[new_workers] 4
mycluster-rhel8-0.example.com
mycluster-rhel8-1.example.com
```

[1] Specify the user name that runs the Ansible tasks on the remote compute machines.

[2] If you do not specify **root** for the **ansible_user**, you must set **ansible_become** to **True** and assign the user sudo permissions.

[3] Specify the path and file name of the **kubeconfig** file for your cluster.

[4] List each RHEL machine to add to your cluster. You must provide the fully-qualified domain name for each host. This name is the hostname that the cluster uses to access the machine, so set the correct public or private name to access the machine.

2. Navigate to the Ansible playbook directory:

   ```
   $ cd /usr/share/ansible/openshift-ansible
   ```

3. Run the playbook:

   ```
   $ ansible-playbook -i /<path>/inventory/hosts playbooks/scaleup.yml 1
   ```

   [1] For **<path>**, specify the path to the Ansible inventory file that you created.

7.1.6. Required parameters for the Ansible hosts file

You must define the following parameters in the Ansible hosts file before you add Red Hat Enterprise Linux (RHEL) compute machines to your cluster.

Parameter	Description	Values
ansible_user	The SSH user that allows SSH-based authentication without requiring a password. If you use SSH key-based authentication, then you must manage the key with an SSH agent.	A user name on the system. The default value is **root**.
ansible_become	If the values of **ansible_user** is not root, you must set **ansible_become** to **True**, and the user that you specify as the **ansible_user** must be configured for passwordless sudo access.	**True**. If the value is not **True**, do not specify and define this parameter.
openshift_kubeconfig_path	Specifies a path and file name to a local directory that contains the **kubeconfig** file for your cluster.	The path and name of the configuration file.

7.1.7. Optional: Removing RHCOS compute machines from a cluster

After you add the Red Hat Enterprise Linux (RHEL) compute machines to your cluster, you can optionally remove the Red Hat Enterprise Linux CoreOS (RHCOS) compute machines to free up resources.

Prerequisites

- You have added RHEL compute machines to your cluster.

Procedure

1. View the list of machines and record the node names of the RHCOS compute machines:

   ```
   $ oc get nodes -o wide
   ```

2. For each RHCOS compute machine, delete the node:

   a. Mark the node as unschedulable by running the **oc adm cordon** command:

      ```
      $ oc adm cordon <node_name>  1
      ```

      1 Specify the node name of one of the RHCOS compute machines.

   b. Drain all the pods from the node:

      ```
      $ oc adm drain <node_name> --force --delete-emptydir-data --ignore-daemonsets  1
      ```

      1 Specify the node name of the RHCOS compute machine that you isolated.

   c. Delete the node:

      ```
      $ oc delete nodes <node_name>  1
      ```

      1 Specify the node name of the RHCOS compute machine that you drained.

3. Review the list of compute machines to ensure that only the RHEL nodes remain:

   ```
   $ oc get nodes -o wide
   ```

4. Remove the RHCOS machines from the load balancer for your cluster's compute machines. You can delete the virtual machines or reimage the physical hardware for the RHCOS compute machines.

7.2. ADDING RHCOS COMPUTE MACHINES TO AN OPENSHIFT CONTAINER PLATFORM CLUSTER

You can add more Red Hat Enterprise Linux CoreOS (RHCOS) compute machines to your OpenShift Container Platform cluster on bare metal.

Before you add more compute machines to a cluster that you installed on bare metal infrastructure, you must create RHCOS machines for it to use. You can either use an ISO image or network PXE booting to create the machines.

7.2.1. Prerequisites

- You installed a cluster on bare metal.

- You have installation media and Red Hat Enterprise Linux CoreOS (RHCOS) images that you used to create your cluster. If you do not have these files, you must obtain them by following the instructions in the installation procedure.

7.2.2. Creating more RHCOS machines using an ISO image

You can create more Red Hat Enterprise Linux CoreOS (RHCOS) compute machines for your bare metal cluster by using an ISO image to create the machines.

Prerequisites

- Obtain the URL of the Ignition config file for the compute machines for your cluster. You uploaded this file to your HTTP server during installation.

Procedure

1. Use the ISO file to install RHCOS on more compute machines. Use the same method that you used when you created machines before you installed the cluster:

   - Burn the ISO image to a disk and boot it directly.

   - Use ISO redirection with a LOM interface.

2. Boot the RHCOS ISO image without specifying any options, or interrupting the live boot sequence. Wait for the installer to boot into a shell prompt in the RHCOS live environment.

   > **NOTE**
   >
   > You can interrupt the RHCOS installation boot process to add kernel arguments. However, for this ISO procedure you must use the **coreos-installer** command as outlined in the following steps, instead of adding kernel arguments.

3. Run the **coreos-installer** command and specify the options that meet your installation requirements. At a minimum, you must specify the URL that points to the Ignition config file for the node type, and the device that you are installing to:

   ```
   $ sudo coreos-installer install --ignition-url=http://<HTTP_server>/<node_type>.ign <device> --ignition-hash=sha512-<digest>  ❶ ❷
   ```

   ❶ You must run the **coreos-installer** command by using **sudo**, because the **core** user does not have the required root privileges to perform the installation.

   ❷ The **--ignition-hash** option is required when the Ignition config file is obtained through an HTTP URL to validate the authenticity of the Ignition config file on the cluster node. **<digest>** is the Ignition config file SHA512 digest obtained in a preceding step.

NOTE

If you want to provide your Ignition config files through an HTTPS server that uses TLS, you can add the internal certificate authority (CA) to the system trust store before running **coreos-installer**.

The following example initializes a bootstrap node installation to the **/dev/sda** device. The Ignition config file for the bootstrap node is obtained from an HTTP web server with the IP address 192.168.1.2:

```
$ sudo coreos-installer install --ignition-
url=http://192.168.1.2:80/installation_directory/bootstrap.ign /dev/sda --ignition-hash=sha512-
a5a2d43879223273c9b60af66b44202a1d1248fc01cf156c46d4a79f552b6bad47bc8cc78ddf011
6e80c59d2ea9e32ba53bc807afbca581aa059311def2c3e3b
```

4. Monitor the progress of the RHCOS installation on the console of the machine.
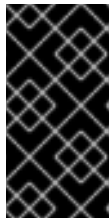
IMPORTANT

Ensure that the installation is successful on each node before commencing with the OpenShift Container Platform installation. Observing the installation process can also help to determine the cause of RHCOS installation issues that might arise.

5. Continue to create more compute machines for your cluster.

7.2.3. Creating more RHCOS machines by PXE or iPXE booting

You can create more Red Hat Enterprise Linux CoreOS (RHCOS) compute machines for your bare metal cluster by using PXE or iPXE booting.

Prerequisites

- Obtain the URL of the Ignition config file for the compute machines for your cluster. You uploaded this file to your HTTP server during installation.

- Obtain the URLs of the RHCOS ISO image, compressed metal BIOS, **kernel**, and **initramfs** files that you uploaded to your HTTP server during cluster installation.

- You have access to the PXE booting infrastructure that you used to create the machines for your OpenShift Container Platform cluster during installation. The machines must boot from their local disks after RHCOS is installed on them.

- If you use UEFI, you have access to the **grub.conf** file that you modified during OpenShift Container Platform installation.

Procedure

1. Confirm that your PXE or iPXE installation for the RHCOS images is correct.

   - For PXE:

     ```
     DEFAULT pxeboot
     TIMEOUT 20
     ```

```
PROMPT 0
LABEL pxeboot
    KERNEL http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture>  1
    APPEND initrd=http://<HTTP_server>/rhcos-<version>-live-initramfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img  2
```

1 Specify the location of the live **kernel** file that you uploaded to your HTTP server.

2 Specify locations of the RHCOS files that you uploaded to your HTTP server. The **initrd** parameter value is the location of the live **initramfs** file, the **coreos.inst.ignition_url** parameter value is the location of the worker Ignition config file, and the **coreos.live.rootfs_url** parameter value is the location of the live **rootfs** file. The **coreos.inst.ignition_url** and **coreos.live.rootfs_url** parameters only support HTTP and HTTPS.

This configuration does not enable serial console access on machines with a graphical console. To configure a different console, add one or more **console=** arguments to the **APPEND** line. For example, add **console=tty0 console=ttyS0** to set the first PC serial port as the primary console and the graphical console as a secondary console. For more information, see How does one set up a serial terminal and/or console in Red Hat Enterprise Linux?.

- For iPXE:

```
kernel http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> initrd=main
coreos.inst.install_dev=/dev/sda coreos.inst.ignition_url=http://<HTTP_server>/worker.ign
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.<architecture>.img
 1
initrd --name main http://<HTTP_server>/rhcos-<version>-live-initramfs.<architecture>.img
 2
```

1 Specify locations of the RHCOS files that you uploaded to your HTTP server. The **kernel** parameter value is the location of the **kernel** file, the **initrd=main** argument is needed for booting on UEFI systems, the **coreos.inst.ignition_url** parameter value is the location of the worker Ignition config file, and the **coreos.live.rootfs_url** parameter value is the location of the live **rootfs** file. The **coreos.inst.ignition_url** and **coreos.live.rootfs_url** parameters only support HTTP and HTTPS.

2 Specify the location of the **initramfs** file that you uploaded to your HTTP server.

This configuration does not enable serial console access on machines with a graphical console. To configure a different console, add one or more **console=** arguments to the **kernel** line. For example, add **console=tty0 console=ttyS0** to set the first PC serial port as the primary console and the graphical console as a secondary console. For more information, see How does one set up a serial terminal and/or console in Red Hat Enterprise Linux?.

1. Use the PXE or iPXE infrastructure to create the required compute machines for your cluster.

7.2.4. Approving the certificate signing requests for your machines

When you add machines to a cluster, two pending certificate signing requests (CSRs) are generated for each machine that you added. You must confirm that these CSRs are approved or, if necessary, approve them yourself. The client requests must be approved first, followed by the server requests.

Prerequisites

- You added machines to your cluster.

Procedure

1. Confirm that the cluster recognizes the machines:

   ```
   $ oc get nodes
   ```

   Example output

   ```
   NAME      STATUS   ROLES  AGE  VERSION
   master-0  Ready    master 63m  v1.25.0
   master-1  Ready    master 63m  v1.25.0
   master-2  Ready    master 64m  v1.25.0
   ```

   The output lists all of the machines that you created.

   > **NOTE**
   >
   > The preceding output might not include the compute nodes, also known as worker nodes, until some CSRs are approved.

2. Review the pending CSRs and ensure that you see the client requests with the **Pending** or **Approved** status for each machine that you added to the cluster:

   ```
   $ oc get csr
   ```

   Example output

   ```
   NAME       AGE   REQUESTOR                                         CONDITION
   csr-8b2br  15m   system:serviceaccount:openshift-machine-config-operator:node-
   bootstrapper   Pending
   csr-8vnps  15m   system:serviceaccount:openshift-machine-config-operator:node-
   bootstrapper   Pending
   ...
   ```

   In this example, two machines are joining the cluster. You might see more approved CSRs in the list.

3. If the CSRs were not approved, after all of the pending CSRs for the machines you added are in **Pending** status, approve the CSRs for your cluster machines:

> **NOTE**
>
> Because the CSRs rotate automatically, approve your CSRs within an hour of adding the machines to the cluster. If you do not approve them within an hour, the certificates will rotate, and more than two certificates will be present for each node. You must approve all of these certificates. After the client CSR is approved, the Kubelet creates a secondary CSR for the serving certificate, which requires manual approval. Then, subsequent serving certificate renewal requests are automatically approved by the **machine-approver** if the Kubelet requests a new certificate with identical parameters.

> **NOTE**
>
> For clusters running on platforms that are not machine API enabled, such as bare metal and other user-provisioned infrastructure, you must implement a method of automatically approving the kubelet serving certificate requests (CSRs). If a request is not approved, then the **oc exec**, **oc rsh**, and **oc logs** commands cannot succeed, because a serving certificate is required when the API server connects to the kubelet. Any operation that contacts the Kubelet endpoint requires this certificate approval to be in place. The method must watch for new CSRs, confirm that the CSR was submitted by the **node-bootstrapper** service account in the **system:node** or **system:admin** groups, and confirm the identity of the node.

- To approve them individually, run the following command for each valid CSR:

  ```
  $ oc adm certificate approve <csr_name> ❶
  ```

  ❶ **<csr_name>** is the name of a CSR from the list of current CSRs.

- To approve all pending CSRs, run the following command:

  ```
  $ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}{{end}}{{end}}' | xargs --no-run-if-empty oc adm certificate approve
  ```

  > **NOTE**
  >
  > Some Operators might not become available until some CSRs are approved.

4. Now that your client requests are approved, you must review the server requests for each machine that you added to the cluster:

   ```
   $ oc get csr
   ```

   Example output

   ```
   NAME        AGE     REQUESTOR                                           CONDITION
   csr-bfd72   5m26s   system:node:ip-10-0-50-126.us-east-2.compute.internal
   Pending
   csr-c57lv   5m26s   system:node:ip-10-0-95-157.us-east-2.compute.internal
   Pending
   ...
   ```

5. If the remaining CSRs are not approved, and are in the **Pending** status, approve the CSRs for your cluster machines:

   - To approve them individually, run the following command for each valid CSR:

     ```
     $ oc adm certificate approve <csr_name>  ❶
     ```

     ❶   **<csr_name>** is the name of a CSR from the list of current CSRs.

   - To approve all pending CSRs, run the following command:

     ```
     $ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
     {{end}}{{end}}' | xargs oc adm certificate approve
     ```

6. After all client and server CSRs have been approved, the machines have the **Ready** status. Verify this by running the following command:

   ```
   $ oc get nodes
   ```

   Example output

   ```
   NAME      STATUS   ROLES   AGE  VERSION
   master-0  Ready    master  73m  v1.25.0
   master-1  Ready    master  73m  v1.25.0
   master-2  Ready    master  74m  v1.25.0
   worker-0  Ready    worker  11m  v1.25.0
   worker-1  Ready    worker  11m  v1.25.0
   ```
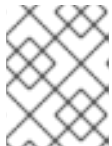
   > **NOTE**
   >
   > It can take a few minutes after approval of the server CSRs for the machines to transition to the **Ready** status.

Additional information

- For more information on CSRs, see Certificate Signing Requests .

7.2.5. Adding a new RHCOS worker node with a custom `/var` partition in AWS

OpenShift Container Platform supports partitioning devices during installation by using machine configs that are processed during the bootstrap. However, if you use **/var** partitioning, the device name must be determined at installation and cannot be changed. You cannot add different instance types as nodes if they have a different device naming schema. For example, if you configured the **/var** partition with the default AWS device name for **m4.large** instances, **dev/xvdb**, you cannot directly add an AWS **m5.large** instance, as **m5.large** instances use a **/dev/nvme1n1** device by default. The device might fail to partition due to the different naming schema.

The procedure in this section shows how to add a new Red Hat Enterprise Linux CoreOS (RHCOS) compute node with an instance that uses a different device name from what was configured at installation. You create a custom user data secret and configure a new compute machine set. These steps are specific to an AWS cluster. The principles apply to other cloud deployments also. However, the device naming schema is different for other deployments and should be determined on a per-case basis.

Procedure

1. On a command line, change to the **openshift-machine-api** namespace:

   ```
   $ oc project openshift-machine-api
   ```

2. Create a new secret from the **worker-user-data** secret:

   a. Export the **userData** section of the secret to a text file:

   ```
   $ oc get secret worker-user-data --template='{{index .data.userData | base64decode}}' | jq > userData.txt
   ```

   b. Edit the text file to add the **storage**, **filesystems**, and **systemd** stanzas for the partitions you want to use for the new node. You can specify any Ignition configuration parameters as needed.

   > **NOTE**
   >
   > Do not change the values in the **ignition** stanza.

   ```
   {
     "ignition": {
       "config": {
         "merge": [
           {
             "source": "https:...."
           }
         ]
       },
       "security": {
        "tls": {
          "certificateAuthorities": [
            {
              "source": "data:text/plain;charset=utf-8;base64,.....=="
            }
          ]
        }
       },
       "version": "3.2.0"
     },
     "storage": {
       "disks": [
         {
           "device": "/dev/nvme1n1",  1
           "partitions": [
             {
               "label": "var",
               "sizeMiB": 50000,  2
               "startMiB": 0  3
             }
           ]
         }
       ],
   ```

```
    "filesystems": [
      {
        "device": "/dev/disk/by-partlabel/var", 4
        "format": "xfs", 5
        "path": "/var" 6
      }
    ]
  },
  "systemd": {
    "units": [ 7
      {
        "contents": "[Unit]\nBefore=local-
fs.target\n[Mount]\nWhere=/var\nWhat=/dev/disk/by-
partlabel/var\nOptions=defaults,pquota\n[Install]\nWantedBy=local-fs.target\n",
        "enabled": true,
        "name": "var.mount"
      }
    ]
  }
}
```

[1] Specifies an absolute path to the AWS block device.

[2] Specifies the size of the data partition in Mebibytes.

[3] Specifies the start of the partition in Mebibytes. When adding a data partition to the boot disk, a minimum value of 25000 MB (Mebibytes) is recommended. The root file system is automatically resized to fill all available space up to the specified offset. If no value is specified, or if the specified value is smaller than the recommended minimum, the resulting root file system will be too small, and future reinstalls of RHCOS might overwrite the beginning of the data partition.

[4] Specifies an absolute path to the **/var** partition.

[5] Specifies the filesystem format.

[6] Specifies the mount-point of the filesystem while Ignition is running relative to where the root filesystem will be mounted. This is not necessarily the same as where it should be mounted in the real root, but it is encouraged to make it the same.

[7] Defines a systemd mount unit that mounts the **/dev/disk/by-partlabel/var** device to the **/var** partition.

c. Extract the **disableTemplating** section from the **work-user-data** secret to a text file:

```
$ oc get secret worker-user-data --template='{{index .data.disableTemplating |
base64decode}}' | jq > disableTemplating.txt
```

d. Create the new user data secret file from the two text files. This user data secret passes the additional node partition information in the **userData.txt** file to the newly created node.

```
$ oc create secret generic worker-user-data-x5 --from-file=userData=userData.txt --
from-file=disableTemplating=disableTemplating.txt
```

3. Create a new compute machine set for the new node:

   a. Create a new compute machine set YAML file, similar to the following, which is configured for AWS. Add the required partitions and the newly-created user data secret:

   TIP

   Use an existing compute machine set as a template and change the parameters as needed for the new node.

   ```
   apiVersion: machine.openshift.io/v1beta1
   kind: MachineSet
   metadata:
     labels:
       machine.openshift.io/cluster-api-cluster: auto-52-92tf4
     name: worker-us-east-2-nvme1n1 ①
     namespace: openshift-machine-api
   spec:
     replicas: 1
     selector:
       matchLabels:
         machine.openshift.io/cluster-api-cluster: auto-52-92tf4
         machine.openshift.io/cluster-api-machineset: auto-52-92tf4-worker-us-east-2b
     template:
       metadata:
         labels:
           machine.openshift.io/cluster-api-cluster: auto-52-92tf4
           machine.openshift.io/cluster-api-machine-role: worker
           machine.openshift.io/cluster-api-machine-type: worker
           machine.openshift.io/cluster-api-machineset: auto-52-92tf4-worker-us-east-2b
       spec:
         metadata: {}
         providerSpec:
           value:
             ami:
               id: ami-0c2dbd95931a
             apiVersion: awsproviderconfig.openshift.io/v1beta1
             blockDevices:
             - DeviceName: /dev/nvme1n1 ②
               ebs:
                 encrypted: true
                 iops: 0
                 volumeSize: 120
                 volumeType: gp2
             - DeviceName: /dev/nvme1n2 ③
               ebs:
                 encrypted: true
                 iops: 0
                 volumeSize: 50
                 volumeType: gp2
             credentialsSecret:
               name: aws-cloud-credentials
             deviceIndex: 0
             iamInstanceProfile:
               id: auto-52-92tf4-worker-profile
   ```

```
      instanceType: m6i.large
      kind: AWSMachineProviderConfig
      metadata:
        creationTimestamp: null
      placement:
        availabilityZone: us-east-2b
        region: us-east-2
      securityGroups:
      - filters:
        - name: tag:Name
          values:
          - auto-52-92tf4-worker-sg
      subnet:
        id: subnet-07a90e5db1
      tags:
      - name: kubernetes.io/cluster/auto-52-92tf4
        value: owned
      userDataSecret:
        name: worker-user-data-x5 4
```

1   Specifies a name for the new node.

2   Specifies an absolute path to the AWS block device, here an encrypted EBS volume.

3   Optional. Specifies an additional EBS volume.

4   Specifies the user data secret file.

    b. Create the compute machine set:

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

    The machines might take a few moments to become available.

4. Verify that the new partition and nodes are created:

    a. Verify that the compute machine set is created:

```
$ oc get machineset
```

Example output

```
NAME                                      DESIRED   CURRENT   READY   AVAILABLE   AGE
ci-ln-2675bt2-76ef8-bdgsc-worker-us-east-1a    1         1         1       1           124m
ci-ln-2675bt2-76ef8-bdgsc-worker-us-east-1b    2         2         2       2           124m
worker-us-east-2-nvme1n1                       1         1         1       1           2m35s 1
```

1   This is the new compute machine set.

    b. Verify that the new node is created:

```
$ oc get nodes
```

Example output

```
NAME                       STATUS  ROLES   AGE     VERSION
ip-10-0-128-78.ec2.internal    Ready    worker   117m   v1.25.0
ip-10-0-146-113.ec2.internal   Ready    master   127m   v1.25.0
ip-10-0-153-35.ec2.internal    Ready    worker   118m   v1.25.0
ip-10-0-176-58.ec2.internal    Ready    master   126m   v1.25.0
ip-10-0-217-135.ec2.internal   Ready    worker   2m57s  v1.25.0 ❶
ip-10-0-225-248.ec2.internal   Ready    master   127m   v1.25.0
ip-10-0-245-59.ec2.internal    Ready    worker   116m   v1.25.0
```

❶ This is new new node.

c. Verify that the custom **/var** partition is created on the new node:

```
$ oc debug node/<node-name> -- chroot /host lsblk
```

For example:

```
$ oc debug node/ip-10-0-217-135.ec2.internal -- chroot /host lsblk
```

Example output

```
NAME        MAJ:MIN  RM  SIZE RO TYPE MOUNTPOINT
nvme0n1     202:0    0   120G 0 disk
|-nvme0n1p1 202:1    0    1M  0 part
|-nvme0n1p2 202:2    0   127M 0 part
|-nvme0n1p3 202:3    0   384M 0 part /boot
`-nvme0n1p4 202:4    0 119.5G 0 part /sysroot
nvme1n1     202:16   0    50G 0 disk
`-nvme1n1p1 202:17   0  48.8G 0 part /var ❶
```

❶ The **nvme1n1** device is mounted to the /var partition.

Additional resources

- For more information on how OpenShift Container Platform uses disk partitioning, see Disk partitioning.

7.3. DEPLOYING MACHINE HEALTH CHECKS

Understand and deploy machine health checks.
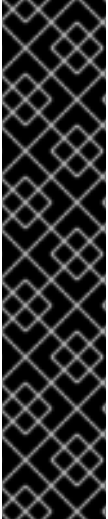
IMPORTANT

You can use the advanced machine management and scaling capabilities only in clusters where the Machine API is operational. Clusters with user-provisioned infrastructure require additional validation and configuration to use the Machine API.

Clusters with the infrastructure platform type **none** cannot use the Machine API. This limitation applies even if the compute machines that are attached to the cluster are installed on a platform that supports the feature. This parameter cannot be changed after installation.

To view the platform type for your cluster, run the following command:

```
$ oc get infrastructure cluster -o jsonpath='{.status.platform}'
```

7.3.1. About machine health checks
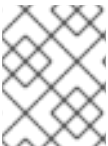
NOTE

You can only apply a machine health check to machines that are managed by compute machine sets or control plane machine sets.

To monitor machine health, create a resource to define the configuration for a controller. Set a condition to check, such as staying in the **NotReady** status for five minutes or displaying a permanent condition in the node-problem-detector, and a label for the set of machines to monitor.

The controller that observes a **MachineHealthCheck** resource checks for the defined condition. If a machine fails the health check, the machine is automatically deleted and one is created to take its place. When a machine is deleted, you see a **machine deleted** event.

To limit disruptive impact of the machine deletion, the controller drains and deletes only one node at a time. If there are more unhealthy machines than the **maxUnhealthy** threshold allows for in the targeted pool of machines, remediation stops and therefore enables manual intervention.

NOTE

Consider the timeouts carefully, accounting for workloads and requirements.

- Long timeouts can result in long periods of downtime for the workload on the unhealthy machine.

- Too short timeouts can result in a remediation loop. For example, the timeout for checking the **NotReady** status must be long enough to allow the machine to complete the startup process.

To stop the check, remove the resource.

7.3.1.1. Limitations when deploying machine health checks

There are limitations to consider before deploying a machine health check:

- Only machines owned by a machine set are remediated by a machine health check.

- If the node for a machine is removed from the cluster, a machine health check considers the machine to be unhealthy and remediates it immediately.

- If the corresponding node for a machine does not join the cluster after the **nodeStartupTimeout**, the machine is remediated.

- A machine is remediated immediately if the **Machine** resource phase is **Failed**.

Additional resources

- [About control plane machine sets](#)

7.3.2. Sample MachineHealthCheck resource

The **MachineHealthCheck** resource for all cloud-based installation types, and other than bare metal, resembles the following YAML file:

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: example 1
  namespace: openshift-machine-api
spec:
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-machine-role: <role> 2
      machine.openshift.io/cluster-api-machine-type: <role> 3
      machine.openshift.io/cluster-api-machineset: <cluster_name>-<label>-<zone> 4
  unhealthyConditions:
  - type:    "Ready"
    timeout: "300s" 5
    status: "False"
  - type:    "Ready"
    timeout: "300s" 6
    status: "Unknown"
  maxUnhealthy: "40%" 7
  nodeStartupTimeout: "10m" 8
```

1	Specify the name of the machine health check to deploy.
2 3	Specify a label for the machine pool that you want to check.
4	Specify the machine set to track in **<cluster_name>-<label>-<zone>** format. For example, **prod-node-us-east-1a**.
5 6	Specify the timeout duration for a node condition. If a condition is met for the duration of the timeout, the machine will be remediated. Long timeouts can result in long periods of downtime for a workload on an unhealthy machine.
7	Specify the amount of machines allowed to be concurrently remediated in the targeted pool. This can be set as a percentage or an integer. If the number of unhealthy machines exceeds the limit set by **maxUnhealthy**, remediation is not performed.
8	Specify the timeout duration that a machine health check must wait for a node to join the cluster before a machine is determined to be unhealthy.

NOTE

The **matchLabels** are examples only; you must map your machine groups based on your specific needs.

7.3.2.1. Short-circuiting machine health check remediation

Short-circuiting ensures that machine health checks remediate machines only when the cluster is healthy. Short-circuiting is configured through the **maxUnhealthy** field in the **MachineHealthCheck** resource.

If the user defines a value for the **maxUnhealthy** field, before remediating any machines, the **MachineHealthCheck** compares the value of **maxUnhealthy** with the number of machines within its target pool that it has determined to be unhealthy. Remediation is not performed if the number of unhealthy machines exceeds the **maxUnhealthy** limit.

IMPORTANT

If **maxUnhealthy** is not set, the value defaults to **100%** and the machines are remediated regardless of the state of the cluster.

The appropriate **maxUnhealthy** value depends on the scale of the cluster you deploy and how many machines the **MachineHealthCheck** covers. For example, you can use the **maxUnhealthy** value to cover multiple compute machine sets across multiple availability zones so that if you lose an entire zone, your **maxUnhealthy** setting prevents further remediation within the cluster. In global Azure regions that do not have multiple availability zones, you can use availability sets to ensure high availability.
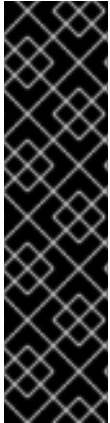
IMPORTANT

If you configure a **MachineHealthCheck** resource for the control plane, set the value of **maxUnhealthy** to **1**.

This configuration ensures that the machine health check takes no action when multiple control plane machines appear to be unhealthy. Multiple unhealthy control plane machines can indicate that the etcd cluster is degraded or that a scaling operation to replace a failed machine is in progress.

If the etcd cluster is degraded, manual intervention might be required. If a scaling operation is in progress, the machine health check should allow it to finish.

The **maxUnhealthy** field can be set as either an integer or percentage. There are different remediation implementations depending on the **maxUnhealthy** value.

7.3.2.1.1. Setting maxUnhealthy by using an absolute value

If **maxUnhealthy** is set to **2**:

- Remediation will be performed if 2 or fewer nodes are unhealthy

- Remediation will not be performed if 3 or more nodes are unhealthy

These values are independent of how many machines are being checked by the machine health check.

7.3.2.1.2. Setting maxUnhealthy by using percentages

If **maxUnhealthy** is set to **40%** and there are 25 machines being checked:

- Remediation will be performed if 10 or fewer nodes are unhealthy

- Remediation will not be performed if 11 or more nodes are unhealthy

If **maxUnhealthy** is set to **40%** and there are 6 machines being checked:

- Remediation will be performed if 2 or fewer nodes are unhealthy

- Remediation will not be performed if 3 or more nodes are unhealthy

> **NOTE**
>
> The allowed number of machines is rounded down when the percentage of **maxUnhealthy** machines that are checked is not a whole number.

7.3.3. Creating a machine health check resource

You can create a **MachineHealthCheck** resource for machine sets in your cluster.

> **NOTE**
>
> You can only apply a machine health check to machines that are managed by compute machine sets or control plane machine sets.

Prerequisites

- Install the **oc** command line interface.

Procedure

1. Create a **healthcheck.yml** file that contains the definition of your machine health check.

2. Apply the **healthcheck.yml** file to your cluster:

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

7.3.4. Scaling a compute machine set manually

To add or remove an instance of a machine in a compute machine set, you can manually scale the compute machine set.

This guidance is relevant to fully automated, installer-provisioned infrastructure installations. Customized, user-provisioned infrastructure installations do not have compute machine sets.

Prerequisites

- Install an OpenShift Container Platform cluster and the **oc** command line.

- Log in to **oc** as a user with **cluster-admin** permission.

Procedure

1. View the compute machine sets that are in the cluster by running the following command:

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

   The compute machine sets are listed in the form of **<clusterid>-worker-<aws-region-az>**.

2. View the compute machines that are in the cluster by running the following command:

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

3. Set the annotation on the compute machine that you want to delete by running the following command:

   ```
   $ oc annotate machine/<machine_name> -n openshift-machine-api
   machine.openshift.io/delete-machine="true"
   ```

4. Scale the compute machine set by running one of the following commands:

   ```
   $ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
   ```

   Or:

   ```
   $ oc edit machineset <machineset> -n openshift-machine-api
   ```

   TIP

   You can alternatively apply the following YAML to scale the compute machine set:

   ```
   apiVersion: machine.openshift.io/v1beta1
   kind: MachineSet
   metadata:
     name: <machineset>
     namespace: openshift-machine-api
   spec:
     replicas: 2
   ```

   You can scale the compute machine set up or down. It takes several minutes for the new machines to be available.

   IMPORTANT

   By default, the machine controller tries to drain the node that is backed by the machine until it succeeds. In some situations, such as with a misconfigured pod disruption budget, the drain operation might not be able to succeed. If the drain operation fails, the machine controller cannot proceed removing the machine.

   You can skip draining the node by annotating **machine.openshift.io/exclude-node-draining** in a specific machine.

Verification

- Verify the deletion of the intended machine by running the following command:

```
$ oc get machines
```

7.3.5. Understanding the difference between compute machine sets and the machine config pool

MachineSet objects describe OpenShift Container Platform nodes with respect to the cloud or machine provider.

The **MachineConfigPool** object allows **MachineConfigController** components to define and provide the status of machines in the context of upgrades.

The **MachineConfigPool** object allows users to configure how upgrades are rolled out to the OpenShift Container Platform nodes in the machine config pool.

The **NodeSelector** object can be replaced with a reference to the **MachineSet** object.

7.4. RECOMMENDED NODE HOST PRACTICES

The OpenShift Container Platform node configuration file contains important options. For example, two parameters control the maximum number of pods that can be scheduled to a node: **podsPerCore** and **maxPods**.

When both options are in use, the lower of the two values limits the number of pods on a node. Exceeding these values can result in:

- Increased CPU utilization.

- Slow pod scheduling.

- Potential out-of-memory scenarios, depending on the amount of memory in the node.

- Exhausting the pool of IP addresses.

- Resource overcommitting, leading to poor user application performance.

> **IMPORTANT**
>
> In Kubernetes, a pod that is holding a single container actually uses two containers. The second container is used to set up networking prior to the actual container starting. Therefore, a system running 10 pods will actually have 20 containers running.

> **NOTE**
>
> Disk IOPS throttling from the cloud provider might have an impact on CRI-O and kubelet. They might get overloaded when there are large number of I/O intensive pods running on the nodes. It is recommended that you monitor the disk I/O on the nodes and use volumes with sufficient throughput for the workload.

The **podsPerCore** parameter sets the number of pods the node can run based on the number of processor cores on the node. For example, if **podsPerCore** is set to **10** on a node with 4 processor cores, the maximum number of pods allowed on the node will be **40**.

```
kubeletConfig:
  podsPerCore: 10
```

■

Setting **podsPerCore** to **0** disables this limit. The default is **0**. The value of the **podsPerCore** parameter cannot exceed the value of the **maxPods** parameter.

The **maxPods** parameter sets the number of pods the node can run to a fixed value, regardless of the properties of the node.

```
kubeletConfig:
  maxPods: 250
```

7.4.1. 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:

```
$ 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 1
...
```

1   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
```

7.4.2. Modifying the number of unavailable worker nodes

By default, only one machine is allowed to be unavailable when applying the kubelet-related configuration to the available worker nodes. For a large cluster, it can take a long time for the configuration change to be reflected. At any time, you can adjust the number of machines that are updating to speed up the process.

Procedure

1. Edit the **worker** machine config pool:

   ```
   $ oc edit machineconfigpool worker
   ```

2. Add the **maxUnavailable** field and set the value:

   ```
   spec:
     maxUnavailable: <node_count>
   ```

> **IMPORTANT**
>
> When setting the value, consider the number of worker nodes that can be unavailable without affecting the applications running on the cluster.

7.4.3. Control plane node sizing

The control plane node resource requirements depend on the number and type of nodes and objects in the cluster. The following control plane node size recommendations are based on the results of a control plane density focused testing, or *Cluster-density*. This test creates the following objects across a given number of namespaces:

- 1 image stream

- 1 build

- 5 deployments, with 2 pod replicas in a **sleep** state, mounting 4 secrets, 4 config maps, and 1 downward API volume each

- 5 services, each one pointing to the TCP/8080 and TCP/8443 ports of one of the previous deployments

- 1 route pointing to the first of the previous services

- 10 secrets containing 2048 random string characters

- 10 config maps containing 2048 random string characters

Number of worker nodes	Cluster-density (namespaces)	CPU cores	Memory (GB)
24	500	4	16
120	1000	8	32
252	4000	16, but 24 if using the OVN-Kubernetes network plug-in	64, but 128 if using the OVN-Kubernetes network plug-in
501, but untested with the OVN-Kubernetes network plug-in	4000	16	96

The data from the table above is based on an OpenShift Container Platform running on top of AWS, using r5.4xlarge instances as control-plane nodes and m5.2xlarge instances as worker nodes.

On a large and dense cluster with three control plane nodes, the CPU and memory usage will spike up when one of the nodes is stopped, rebooted, or fails. The failures can be due to unexpected issues with power, network, underlying infrastructure, or intentional cases where the cluster is restarted after shutting it down to save costs. The remaining two control plane nodes must handle the load in order to be highly available, which leads to increase in the resource usage. This is also expected during upgrades because the control plane nodes are cordoned, drained, and rebooted serially to apply the operating system updates, as well as the control plane Operators update. To avoid cascading failures, keep the overall CPU and memory resource usage on the control plane nodes to at most 60% of all available capacity to handle the resource usage spikes. Increase the CPU and memory on the control plane nodes accordingly to avoid potential downtime due to lack of resources.

> **IMPORTANT**
>
> The node sizing varies depending on the number of nodes and object counts in the cluster. It also depends on whether the objects are actively being created on the cluster. During object creation, the control plane is more active in terms of resource usage compared to when the objects are in the **running** phase.

Operator Lifecycle Manager (OLM) runs on the control plane nodes and its memory footprint depends on the number of namespaces and user installed operators that OLM needs to manage on the cluster. Control plane nodes need to be sized accordingly to avoid OOM kills. Following data points are based on the results from cluster maximums testing.

Number of namespaces	OLM memory at idle state (GB)	OLM memory with 5 user operators installed (GB)
500	0.823	1.7
1000	1.2	2.5
1500	1.7	3.2

Number of namespaces	OLM memory at idle state (GB)	OLM memory with 5 user operators installed (GB)
2000	2	4.4
3000	2.7	5.6
4000	3.8	7.6
5000	4.2	9.02
6000	5.8	11.3
7000	6.6	12.9
8000	6.9	14.8
9000	8	17.7
10,000	9.9	21.6
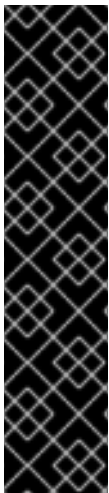
IMPORTANT

You can modify the control plane node size in a running OpenShift Container Platform 4.12 cluster for the following configurations only:

- Clusters installed with a user-provisioned installation method.

- AWS clusters installed with an installer-provisioned infrastructure installation method.

- Clusters that use a control plane machine set to manage control plane machines.

For all other configurations, you must estimate your total node count and use the suggested control plane node size during installation.
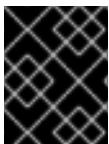
IMPORTANT

The recommendations are based on the data points captured on OpenShift Container Platform clusters with OpenShift SDN as the network plugin.

NOTE

In OpenShift Container Platform 4.12, half of a CPU core (500 millicore) is now reserved by the system by default compared to OpenShift Container Platform 3.11 and previous versions. The sizes are determined taking that into consideration.

7.4.4. Setting up CPU Manager

Procedure

Procedure

1. Optional: Label a node:

   ```
   # oc label node perf-node.example.com cpumanager=true
   ```

2. Edit the **MachineConfigPool** of the nodes where CPU Manager should be enabled. In this example, all workers have CPU Manager enabled:

   ```
   # oc edit machineconfigpool worker
   ```

3. Add a label to the worker machine config pool:

   ```
   metadata:
     creationTimestamp: 2020-xx-xxx
     generation: 3
     labels:
       custom-kubelet: cpumanager-enabled
   ```

4. Create a **KubeletConfig**, **cpumanager-kubeletconfig.yaml**, custom resource (CR). Refer to the label created in the previous step to have the correct nodes updated with the new kubelet config. See the **machineConfigPoolSelector** section:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: KubeletConfig
   metadata:
     name: cpumanager-enabled
   spec:
     machineConfigPoolSelector:
       matchLabels:
         custom-kubelet: cpumanager-enabled
     kubeletConfig:
       cpuManagerPolicy: static         ❶
       cpuManagerReconcilePeriod: 5s    ❷
   ```

   ❶ Specify a policy:

   - **none**. This policy explicitly enables the existing default CPU affinity scheme, providing no affinity beyond what the scheduler does automatically. This is the default policy.

   - **static**. This policy allows containers in guaranteed pods with integer CPU requests. It also limits access to exclusive CPUs on the node. If **static**, you must use a lowercase **s**.

   ❷ Optional. Specify the CPU Manager reconcile frequency. The default is **5s**.

5. Create the dynamic kubelet config:

   ```
   # oc create -f cpumanager-kubeletconfig.yaml
   ```

   This adds the CPU Manager feature to the kubelet config and, if needed, the Machine Config Operator (MCO) reboots the node. To enable CPU Manager, a reboot is not needed.

6. Check for the merged kubelet config:

```
# oc get machineconfig 99-worker-XXXXXX-XXXXX-XXXX-XXXXX-kubelet -o json | grep
ownerReference -A7
```

Example output

```
"ownerReferences": [
  {
    "apiVersion": "machineconfiguration.openshift.io/v1",
    "kind": "KubeletConfig",
    "name": "cpumanager-enabled",
    "uid": "7ed5616d-6b72-11e9-aae1-021e1ce18878"
  }
]
```

7. Check the worker for the updated **kubelet.conf**:

```
# oc debug node/perf-node.example.com
sh-4.2# cat /host/etc/kubernetes/kubelet.conf | grep cpuManager
```

Example output

```
cpuManagerPolicy: static          1
cpuManagerReconcilePeriod: 5s     2
```

1   **cpuManagerPolicy** is defined when you create the **KubeletConfig** CR.

2   **cpuManagerReconcilePeriod** is defined when you create the **KubeletConfig** CR.

8. Create a pod that requests a core or multiple cores. Both limits and requests must have their CPU value set to a whole integer. That is the number of cores that will be dedicated to this pod:

```
# cat cpumanager-pod.yaml
```

Example output

```
apiVersion: v1
kind: Pod
metadata:
  generateName: cpumanager-
spec:
  containers:
  - name: cpumanager
    image: gcr.io/google_containers/pause-amd64:3.0
    resources:
      requests:
        cpu: 1
        memory: "1G"
      limits:
        cpu: 1
        memory: "1G"
  nodeSelector:
    cpumanager: "true"
```

9. Create the pod:

```
# oc create -f cpumanager-pod.yaml
```

10. Verify that the pod is scheduled to the node that you labeled:

```
# oc describe pod cpumanager
```

Example output

```
Name:          cpumanager-6cqz7
Namespace:        default
Priority:        0
PriorityClassName:  <none>
Node:  perf-node.example.com/xxx.xx.xx.xxx
...
 Limits:
    cpu:     1
    memory:  1G
   Requests:
    cpu:      1
    memory:    1G
...
QoS Class:      Guaranteed
Node-Selectors:  cpumanager=true
```

11. Verify that the **cgroups** are set up correctly. Get the process ID (PID) of the **pause** process:

```
#  ├──init.scope
|  └──1 /usr/lib/systemd/systemd --switched-root --system --deserialize 17
└──kubepods.slice
  ├──kubepods-pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice
  │  ├──crio-b5437308f1a574c542bdf08563b865c0345c8f8c0b0a655612c.scope
  │  └──32706 /pause
```

Pods of quality of service (QoS) tier **Guaranteed** are placed within the **kubepods.slice**. Pods of other QoS tiers end up in child **cgroups** of **kubepods**:

```
# cd /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-
pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice/crio-
b5437308f1ad1a7db0574c542bdf08563b865c0345c86e9585f8c0b0a655612c.scope
# for i in `ls cpuset.cpus tasks` ; do echo -n "$i "; cat $i ; done
```

Example output

```
cpuset.cpus 1
tasks 32706
```

12. Check the allowed CPU list for the task:

```
# grep ^Cpus_allowed_list /proc/32706/status
```

Example output

> Cpus_allowed_list: 1

13. Verify that another pod (in this case, the pod in the **burstable** QoS tier) on the system cannot run on the core allocated for the **Guaranteed** pod:

```
# cat /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-besteffort.slice/kubepods-besteffort-podc494a073_6b77_11e9_98c0_06bba5c387ea.slice/crio-c56982f57b75a2420947f0afc6cafe7534c5734efc34157525fa9abbf99e3849.scope/cpuset.cpus

0
# oc describe node perf-node.example.com
```

Example output

```
...
Capacity:
 attachable-volumes-aws-ebs:  39
 cpu:                    2
 ephemeral-storage:          124768236Ki
 hugepages-1Gi:              0
 hugepages-2Mi:              0
 memory:             8162900Ki
 pods:               250
Allocatable:
 attachable-volumes-aws-ebs:  39
 cpu:                1500m
 ephemeral-storage:          124768236Ki
 hugepages-1Gi:              0
 hugepages-2Mi:              0
 memory:             7548500Ki
 pods:               250
-------                      ----                     ------------ ---------- --------------- ------------- --
-
  default                        cpumanager-6cqz7          1 (66%)      1 (66%)    1G (12%)
1G (12%)       29m

Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource                Requests        Limits
  --------                --------        ------
  cpu                 1440m (96%)      1 (66%)
```

This VM has two CPU cores. The **system-reserved** setting reserves 500 millicores, meaning that half of one core is subtracted from the total capacity of the node to arrive at the **Node Allocatable** amount. You can see that **Allocatable CPU** is 1500 millicores. This means you can run one of the CPU Manager pods since each will take one whole core. A whole core is equivalent to 1000 millicores. If you try to schedule a second pod, the system will accept the pod, but it will never be scheduled:

```
NAME                READY  STATUS   RESTARTS  AGE
cpumanager-6cqz7      1/1    Running  0         33m
cpumanager-7qc2t      0/1    Pending  0         11s
```

7.5. HUGE PAGES

Understand and configure huge pages.

7.5.1. What huge pages do

Memory is managed in blocks known as pages. On most systems, a page is 4Ki. 1Mi of memory is equal to 256 pages; 1Gi of memory is 256,000 pages, and so on. CPUs have a built-in memory management unit that manages a list of these pages in hardware. The Translation Lookaside Buffer (TLB) is a small hardware cache of virtual-to-physical page mappings. If the virtual address passed in a hardware instruction can be found in the TLB, the mapping can be determined quickly. If not, a TLB miss occurs, and the system falls back to slower, software-based address translation, resulting in performance issues. Since the size of the TLB is fixed, the only way to reduce the chance of a TLB miss is to increase the page size.

A huge page is a memory page that is larger than 4Ki. On x86_64 architectures, there are two common huge page sizes: 2Mi and 1Gi. Sizes vary on other architectures. To use huge pages, code must be written so that applications are aware of them. Transparent Huge Pages (THP) attempt to automate the management of huge pages without application knowledge, but they have limitations. In particular, they are limited to 2Mi page sizes. THP can lead to performance degradation on nodes with high memory utilization or fragmentation due to defragmenting efforts of THP, which can lock memory pages. For this reason, some applications may be designed to (or recommend) usage of pre-allocated huge pages instead of THP.

7.5.2. How huge pages are consumed by apps

Nodes must pre-allocate huge pages in order for the node to report its huge page capacity. A node can only pre-allocate huge pages for a single size.

Huge pages can be consumed through container-level resource requirements using the resource name **hugepages-<size>**, where size is the most compact binary notation using integer values supported on a particular node. For example, if a node supports 2048KiB page sizes, it exposes a schedulable resource **hugepages-2Mi**. Unlike CPU or memory, huge pages do not support over-commitment.

```
apiVersion: v1
kind: Pod
metadata:
  generateName: hugepages-volume-
spec:
  containers:
  - securityContext:
      privileged: true
    image: rhel7:latest
    command:
    - sleep
    - inf
    name: example
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
        hugepages-2Mi: 100Mi  1
        memory: "1Gi"
        cpu: "1"
```

```
volumes:
- name: hugepage
  emptyDir:
    medium: HugePages
```

[1] Specify the amount of memory for **hugepages** as the exact amount to be allocated. Do not specify this value as the amount of memory for **hugepages** multiplied by the size of the page. For example, given a huge page size of 2MB, if you want to use 100MB of huge-page-backed RAM for your application, then you would allocate 50 huge pages. OpenShift Container Platform handles the math for you. As in the above example, you can specify **100MB** directly.

Allocating huge pages of a specific size

Some platforms support multiple huge page sizes. To allocate huge pages of a specific size, precede the huge pages boot command parameters with a huge page size selection parameter **hugepagesz=<size>**. The **<size>** value must be specified in bytes with an optional scale suffix [**kKmMgG**]. The default huge page size can be defined with the **default_hugepagesz=<size>** boot parameter.

Huge page requirements

- Huge page requests must equal the limits. This is the default if limits are specified, but requests are not.

- Huge pages are isolated at a pod scope. Container isolation is planned in a future iteration.

- **EmptyDir** volumes backed by huge pages must not consume more huge page memory than the pod request.

- Applications that consume huge pages via **shmget()** with **SHM_HUGETLB** must run with a supplemental group that matches *proc/sys/vm/hugetlb_shm_group*.

7.5.3. Configuring huge pages at boot time

Nodes must pre-allocate huge pages used in an OpenShift Container Platform cluster. There are two ways of reserving huge pages: at boot time and at run time. Reserving at boot time increases the possibility of success because the memory has not yet been significantly fragmented. The Node Tuning Operator currently supports boot time allocation of huge pages on specific nodes.

Procedure

To minimize node reboots, the order of the steps below needs to be followed:

1. Label all nodes that need the same huge pages setting by a label.

   ```
   $ oc label node <node_using_hugepages> node-role.kubernetes.io/worker-hp=
   ```

2. Create a file with the following content and name it **hugepages-tuned-boottime.yaml**:

   ```
   apiVersion: tuned.openshift.io/v1
   kind: Tuned
   metadata:
     name: hugepages [1]
     namespace: openshift-cluster-node-tuning-operator
   spec:
     profile: [2]
   ```

```
- data: |
    [main]
    summary=Boot time configuration for hugepages
    include=openshift-node
    [bootloader]
    cmdline_openshift_node_hugepages=hugepagesz=2M hugepages=50
  name: openshift-node-hugepages

recommend:
- machineConfigLabels:
    machineconfiguration.openshift.io/role: "worker-hp"
  priority: 30
  profile: openshift-node-hugepages
```
3

4

1 Set the **name** of the Tuned resource to **hugepages**.

2 Set the **profile** section to allocate huge pages.

3 Note the order of parameters is important as some platforms support huge pages of various sizes.

4 Enable machine config pool based matching.

3. Create the Tuned **hugepages** object

   ```
   $ oc create -f hugepages-tuned-boottime.yaml
   ```

4. Create a file with the following content and name it **hugepages-mcp.yaml**:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfigPool
   metadata:
     name: worker-hp
     labels:
       worker-hp: ""
   spec:
     machineConfigSelector:
       matchExpressions:
         - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,worker-hp]}
     nodeSelector:
       matchLabels:
         node-role.kubernetes.io/worker-hp: ""
   ```

5. Create the machine config pool:

   ```
   $ oc create -f hugepages-mcp.yaml
   ```

Given enough non-fragmented memory, all the nodes in the **worker-hp** machine config pool should now have 50 2Mi huge pages allocated.

```
$ oc get node <node_using_hugepages> -o jsonpath="{.status.allocatable.hugepages-2Mi}"
100Mi
```

> **NOTE**
>
> The TuneD bootloader plugin only supports Red Hat Enterprise Linux CoreOS (RHCOS) worker nodes.

7.6. UNDERSTANDING DEVICE PLUGINS

The device plugin provides a consistent and portable solution to consume hardware devices across clusters. The device plugin provides support for these devices through an extension mechanism, which makes these devices available to Containers, provides health checks of these devices, and securely shares them.
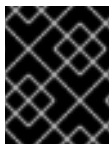
> **IMPORTANT**
>
> OpenShift Container Platform supports the device plugin API, but the device plugin Containers are supported by individual vendors.

A device plugin is a gRPC service running on the nodes (external to the **kubelet**) that is responsible for managing specific hardware resources. Any device plugin must support following remote procedure calls (RPCs):

```
service DevicePlugin {
      // GetDevicePluginOptions returns options to be communicated with Device
      // Manager
      rpc GetDevicePluginOptions(Empty) returns (DevicePluginOptions) {}

      // ListAndWatch returns a stream of List of Devices
      // Whenever a Device state change or a Device disappears, ListAndWatch
      // returns the new list
      rpc ListAndWatch(Empty) returns (stream ListAndWatchResponse) {}

      // Allocate is called during container creation so that the Device
      // Plug-in can run device specific operations and instruct Kubelet
      // of the steps to make the Device available in the container
      rpc Allocate(AllocateRequest) returns (AllocateResponse) {}

      // PreStartcontainer is called, if indicated by Device Plug-in during
      // registration phase, before each container start. Device plug-in
      // can run device specific operations such as resetting the device
      // before making devices available to the container
      rpc PreStartcontainer(PreStartcontainerRequest) returns (PreStartcontainerResponse) {}
}
```

Example device plugins

- [Nvidia GPU device plugin for COS-based operating system](#)

- [Nvidia official GPU device plugin](#)

- [Solarflare device plugin](#)

- [KubeVirt device plugins: vfio and kvm](#)

- [Kubernetes device plugin for IBM Crypto Express (CEX) cards](#)

> **NOTE**
>
> For easy device plugin reference implementation, there is a stub device plugin in the Device Manager code: *vendor/k8s.io/kubernetes/pkg/kubelet/cm/deviceplugin/device_plugin_stub.go*.

7.6.1. Methods for deploying a device plugin

- Daemon sets are the recommended approach for device plugin deployments.

- Upon start, the device plugin will try to create a UNIX domain socket at */var/lib/kubelet/device-plugin/* on the node to serve RPCs from Device Manager.

- Since device plugins must manage hardware resources, access to the host file system, as well as socket creation, they must be run in a privileged security context.

- More specific details regarding deployment steps can be found with each device plugin implementation.

7.6.2. Understanding the Device Manager

Device Manager provides a mechanism for advertising specialized node hardware resources with the help of plugins known as device plugins.

You can advertise specialized hardware without requiring any upstream code changes.

> **IMPORTANT**
>
> OpenShift Container Platform supports the device plugin API, but the device plugin Containers are supported by individual vendors.

Device Manager advertises devices as **Extended Resources**. User pods can consume devices, advertised by Device Manager, using the same **Limit/Request** mechanism, which is used for requesting any other **Extended Resource**.

Upon start, the device plugin registers itself with Device Manager invoking **Register** on the */var/lib/kubelet/device-plugins/kubelet.sock* and starts a gRPC service at */var/lib/kubelet/device-plugins/<plugin>.sock* for serving Device Manager requests.

Device Manager, while processing a new registration request, invokes **ListAndWatch** remote procedure call (RPC) at the device plugin service. In response, Device Manager gets a list of **Device** objects from the plugin over a gRPC stream. Device Manager will keep watching on the stream for new updates from the plugin. On the plugin side, the plugin will also keep the stream open and whenever there is a change in the state of any of the devices, a new device list is sent to the Device Manager over the same streaming connection.

While handling a new pod admission request, Kubelet passes requested **Extended Resources** to the Device Manager for device allocation. Device Manager checks in its database to verify if a corresponding plugin exists or not. If the plugin exists and there are free allocatable devices as well as per local cache, **Allocate** RPC is invoked at that particular device plugin.

Additionally, device plugins can also perform several other device-specific operations, such as driver installation, device initialization, and device resets. These functionalities vary from implementation to implementation.

7.6.3. Enabling Device Manager

Enable Device Manager to implement a device plugin to advertise specialized hardware without any upstream code changes.

Device Manager provides a mechanism for advertising specialized node hardware resources with the help of plugins known as device plugins.

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

    a. View the machine config:

    ```
    # oc describe machineconfig <name>
    ```

    For example:

    ```
    # oc describe machineconfig 00-worker
    ```

    Example output

    ```
    Name:        00-worker
    Namespace:
    Labels:      machineconfiguration.openshift.io/role=worker  1
    ```

    1   Label required for the Device Manager.

Procedure

1. Create a custom resource (CR) for your configuration change.

    Sample configuration for a Device Manager CR

    ```
    apiVersion: machineconfiguration.openshift.io/v1
    kind: KubeletConfig
    metadata:
      name: devicemgr  1
    spec:
      machineConfigPoolSelector:
        matchLabels:
          machineconfiguration.openshift.io: devicemgr  2
      kubeletConfig:
        feature-gates:
          - DevicePlugins=true  3
    ```

    1   Assign a name to CR.

    2   Enter the label from the Machine Config Pool.

    3   Set **DevicePlugins** to 'true`.

2. Create the Device Manager:

```
$ oc create -f devicemgr.yaml
```

Example output

```
kubeletconfig.machineconfiguration.openshift.io/devicemgr created
```

3. Ensure that Device Manager was actually enabled by confirming that */var/lib/kubelet/device-plugins/kubelet.sock* is created on the node. This is the UNIX domain socket on which the Device Manager gRPC server listens for new plugin registrations. This sock file is created when the Kubelet is started only if Device Manager is enabled.

7.7. TAINTS AND TOLERATIONS

Understand and work with taints and tolerations.

7.7.1. Understanding taints and tolerations

A *taint* allows a node to refuse a pod to be scheduled unless that pod has a matching *toleration*.

You apply taints to a node through the **Node** specification (**NodeSpec**) and apply tolerations to a pod through the **Pod** specification (**PodSpec**). When you apply a taint a node, the scheduler cannot place a pod on that node unless the pod can tolerate the taint.

Example taint in a node specification

```
apiVersion: v1
kind: Node
metadata:
  name: my-node
#...
spec:
  taints:
  - effect: NoExecute
    key: key1
    value: value1
#...
```

Example toleration in a Pod spec

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
    tolerationSeconds: 3600
#...
```

Taints and tolerations consist of a key, value, and effect.

Table 7.1. Taint and toleration components

Parameter	Description
key	The **key** is any string, up to 253 characters. The key must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores.
value	The **value** is any string, up to 63 characters. The value must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores.
effect	The effect is one of the following: <table><tr><td>**NoSchedule** [1]</td><td> - New pods that do not match the taint are not scheduled onto that node. - Existing pods on the node remain. </td></tr><tr><td>**PreferNoSchedule**</td><td> - New pods that do not match the taint might be scheduled onto that node, but the scheduler tries not to. - Existing pods on the node remain. </td></tr><tr><td>**NoExecute**</td><td> - New pods that do not match the taint cannot be scheduled onto that node. - Existing pods on the node that do not have a matching toleration are removed. </td></tr></table>
operator	<table><tr><td>**Equal**</td><td>The **key**/**value**/**effect** parameters must match. This is the default.</td></tr><tr><td>**Exists**</td><td>The **key**/**effect** parameters must match. You must leave a blank **value** parameter, which matches any.</td></tr></table>

1. If you add a **NoSchedule** taint to a control plane node, the node must have the **node-role.kubernetes.io/master=:NoSchedule** taint, which is added by default.
   For example:

   ```
   apiVersion: v1
   kind: Node
   metadata:
     annotations:
       machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-f76d1-v8jxv-master-0
   ```

```
    machineconfiguration.openshift.io/currentConfig: rendered-master-
  cdc1ab7da414629332cc4c3926e6e59c
   name: my-node
 #...
 spec:
   taints:
   - effect: NoSchedule
     key: node-role.kubernetes.io/master
   #...
```

A toleration matches a taint:

- If the **operator** parameter is set to **Equal**:

  - the **key** parameters are the same;

  - the **value** parameters are the same;

  - the **effect** parameters are the same.

- If the **operator** parameter is set to **Exists**:

  - the **key** parameters are the same;

  - the **effect** parameters are the same.

The following taints are built into OpenShift Container Platform:

- **node.kubernetes.io/not-ready**: The node is not ready. This corresponds to the node condition **Ready=False**.

- **node.kubernetes.io/unreachable**: The node is unreachable from the node controller. This corresponds to the node condition **Ready=Unknown**.

- **node.kubernetes.io/memory-pressure**: The node has memory pressure issues. This corresponds to the node condition **MemoryPressure=True**.

- **node.kubernetes.io/disk-pressure**: The node has disk pressure issues. This corresponds to the node condition **DiskPressure=True**.

- **node.kubernetes.io/network-unavailable**: The node network is unavailable.

- **node.kubernetes.io/unschedulable**: The node is unschedulable.

- **node.cloudprovider.kubernetes.io/uninitialized**: When the node controller is started with an external cloud provider, this taint is set on a node to mark it as unusable. After a controller from the cloud-controller-manager initializes this node, the kubelet removes this taint.

- **node.kubernetes.io/pid-pressure**: The node has pid pressure. This corresponds to the node condition **PIDPressure=True**.
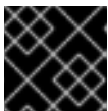
> **IMPORTANT**
>
> OpenShift Container Platform does not set a default pid.available **evictionHard**.

7.7.2. Adding taints and tolerations

You add tolerations to pods and taints to nodes to allow the node to control which pods should or should not be scheduled on them. For existing pods and nodes, you should add the toleration to the pod first, then add the taint to the node to avoid pods being removed from the node before you can add the toleration.

Procedure

1. Add a toleration to a pod by editing the **Pod** spec to include a **tolerations** stanza:

   Sample pod configuration file with an Equal operator

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: my-pod
   #...
   spec:
     tolerations:
     - key: "key1" 1
       value: "value1"
       operator: "Equal"
       effect: "NoExecute"
       tolerationSeconds: 3600 2
   #...
   ```

   1 The toleration parameters, as described in the **Taint and toleration components** table.

   2 The **tolerationSeconds** parameter specifies how long a pod can remain bound to a node before being evicted.

   For example:

   Sample pod configuration file with an Exists operator

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: my-pod
   #...
   spec:
     tolerations:
      - key: "key1"
        operator: "Exists" 1
        effect: "NoExecute"
        tolerationSeconds: 3600
   #...
   ```

   1 The **Exists** operator does not take a **value**.

   This example places a taint on **node1** that has key **key1**, value **value1**, and taint effect **NoExecute**.

2. Add a taint to a node by using the following command with the parameters described in the **Taint and toleration components** table:

```
$ oc adm taint nodes <node_name> <key>=<value>:<effect>
```

For example:

```
$ oc adm taint nodes node1 key1=value1:NoExecute
```

This command places a taint on **node1** that has key **key1**, value **value1**, and effect **NoExecute**.

> **NOTE**
>
> If you add a **NoSchedule** taint to a control plane node, the node must have the **node-role.kubernetes.io/master=:NoSchedule** taint, which is added by default.
>
> For example:
>
> ```
> apiVersion: v1
> kind: Node
> metadata:
>   annotations:
>     machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-f76d1-
> v8jxv-master-0
>     machineconfiguration.openshift.io/currentConfig: rendered-master-
> cdc1ab7da414629332cc4c3926e6e59c
>   name: my-node
> #...
> spec:
>   taints:
>   - effect: NoSchedule
>     key: node-role.kubernetes.io/master
> #...
> ```

The tolerations on the pod match the taint on the node. A pod with either toleration can be scheduled onto **node1**.

7.7.3. Adding taints and tolerations using a compute machine set

You can add taints to nodes using a compute machine set. All nodes associated with the **MachineSet** object are updated with the taint. Tolerations respond to taints added by a compute machine set in the same manner as taints added directly to the nodes.

Procedure

1. Add a toleration to a pod by editing the **Pod** spec to include a **tolerations** stanza:

   Sample pod configuration file with Equal operator

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: my-pod
   ```

```
#...
spec:
 tolerations:
 - key: "key1"  ❶
   value: "value1"
   operator: "Equal"
   effect: "NoExecute"
   tolerationSeconds: 3600  ❷
#...
```

❶   The toleration parameters, as described in the **Taint and toleration components** table.

❷   The **tolerationSeconds** parameter specifies how long a pod is bound to a node before being evicted.

For example:

Sample pod configuration file with Exists operator

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
 tolerations:
 - key: "key1"
   operator: "Exists"
   effect: "NoExecute"
   tolerationSeconds: 3600
#...
```

2. Add the taint to the **MachineSet** object:

   a. Edit the **MachineSet** YAML for the nodes you want to taint or you can create a new **MachineSet** object:

      ```
      $ oc edit machineset <machineset>
      ```

   b. Add the taint to the **spec.template.spec** section:

      Example taint in a compute machine set specification

      ```
      apiVersion: machine.openshift.io/v1beta1
      kind: MachineSet
      metadata:
        name: my-machineset
      #...
      spec:
      #...
        template:
      #...
          spec:
            taints:
      ```

```
      - effect: NoExecute
        key: key1
        value: value1
#...
```

This example places a taint that has the key **key1**, value **value1**, and taint effect **NoExecute** on the nodes.

c. Scale down the compute machine set to 0:

```
$ oc scale --replicas=0 machineset <machineset> -n openshift-machine-api
```

TIP

You can alternatively apply the following YAML to scale the compute machine set:

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 0
```

Wait for the machines to be removed.

d. Scale up the compute machine set as needed:

```
$ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
```

Or:

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

Wait for the machines to start. The taint is added to the nodes associated with the **MachineSet** object.

7.7.4. Binding a user to a node using taints and tolerations

If you want to dedicate a set of nodes for exclusive use by a particular set of users, add a toleration to their pods. Then, add a corresponding taint to those nodes. The pods with the tolerations are allowed to use the tainted nodes or any other nodes in the cluster.

If you want ensure the pods are scheduled to only those tainted nodes, also add a label to the same set of nodes and add a node affinity to the pods so that the pods can only be scheduled onto nodes with that label.

Procedure

To configure a node so that users can use only that node:

1. Add a corresponding taint to those nodes:
   For example:

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

TIP

You can alternatively apply the following YAML to add the taint:

```
kind: Node
apiVersion: v1
metadata:
  name: my-node
#...
spec:
  taints:
    - key: dedicated
      value: groupName
      effect: NoSchedule
#...
```

2. Add a toleration to the pods by writing a custom admission controller.

7.7.5. Controlling nodes with special hardware using taints and tolerations

In a cluster where a small subset of nodes have specialized hardware, you can use taints and tolerations to keep pods that do not need the specialized hardware off of those nodes, leaving the nodes for pods that do need the specialized hardware. You can also require pods that need specialized hardware to use specific nodes.

You can achieve this by adding a toleration to pods that need the special hardware and tainting the nodes that have the specialized hardware.

Procedure

To ensure nodes with specialized hardware are reserved for specific pods:

1. Add a toleration to pods that need the special hardware.
   For example:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
    - key: "disktype"
      value: "ssd"
      operator: "Equal"
      effect: "NoSchedule"
      tolerationSeconds: 3600
#...
```

2. Taint the nodes that have the specialized hardware using one of the following commands:

```
$ oc adm taint nodes <node-name> disktype=ssd:NoSchedule
```

Or:

```
$ oc adm taint nodes <node-name> disktype=ssd:PreferNoSchedule
```

TIP

You can alternatively apply the following YAML to add the taint:

```
kind: Node
apiVersion: v1
metadata:
  name: my_node
#...
spec:
 taints:
   - key: disktype
     value: ssd
     effect: PreferNoSchedule
#...
```

7.7.6. Removing taints and tolerations

You can remove taints from nodes and tolerations from pods as needed. You should add the toleration to the pod first, then add the taint to the node to avoid pods being removed from the node before you can add the toleration.

Procedure

To remove taints and tolerations:

1. To remove a taint from a node:

   ```
   $ oc adm taint nodes <node-name> <key>-
   ```

   For example:

   ```
   $ oc adm taint nodes ip-10-0-132-248.ec2.internal key1-
   ```

   Example output

   ```
   node/ip-10-0-132-248.ec2.internal untainted
   ```

2. To remove a toleration from a pod, edit the **Pod** spec to remove the toleration:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: my-pod
   #...
   spec:
    tolerations:
    - key: "key2"
   ```

```
        operator: "Exists"
        effect: "NoExecute"
        tolerationSeconds: 3600
    #...
```

7.8. TOPOLOGY MANAGER

Understand and work with Topology Manager.

7.8.1. Topology Manager policies

Topology Manager aligns **Pod** resources of all Quality of Service (QoS) classes by collecting topology hints from Hint Providers, such as CPU Manager and Device Manager, and using the collected hints to align the **Pod** resources.

Topology Manager supports four allocation policies, which you assign in the **KubeletConfig** custom resource (CR) named **cpumanager-enabled**:

none policy

This is the default policy and does not perform any topology alignment.

best-effort policy

For each container in a pod with the **best-effort** topology management policy, kubelet calls each Hint Provider to discover their resource availability. Using this information, the Topology Manager stores the preferred NUMA Node affinity for that container. If the affinity is not preferred, Topology Manager stores this and admits the pod to the node.

restricted policy

For each container in a pod with the **restricted** topology management policy, kubelet calls each Hint Provider to discover their resource availability. Using this information, the Topology Manager stores the preferred NUMA Node affinity for that container. If the affinity is not preferred, Topology Manager rejects this pod from the node, resulting in a pod in a **Terminated** state with a pod admission failure.

single-numa-node policy

For each container in a pod with the **single-numa-node** topology management policy, kubelet calls each Hint Provider to discover their resource availability. Using this information, the Topology Manager determines if a single NUMA Node affinity is possible. If it is, the pod is admitted to the node. If a single NUMA Node affinity is not possible, the Topology Manager rejects the pod from the node. This results in a pod in a Terminated state with a pod admission failure.

7.8.2. Setting up Topology Manager

To use Topology Manager, you must configure an allocation policy in the **KubeletConfig** custom resource (CR) named **cpumanager-enabled**. This file might exist if you have set up CPU Manager. If the file does not exist, you can create the file.

Prerequisites

- Configure the CPU Manager policy to be **static**.

Procedure

To activate Topology Manager:

1. Configure the Topology Manager allocation policy in the custom resource.

```
$ oc edit KubeletConfig cpumanager-enabled
```

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cpumanager-enabled
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: cpumanager-enabled
  kubeletConfig:
    cpuManagerPolicy: static 1
    cpuManagerReconcilePeriod: 5s
    topologyManagerPolicy: single-numa-node 2
```

1 This parameter must be **static** with a lowercase **s**.

2 Specify your selected Topology Manager allocation policy. Here, the policy is **single-numa-node**. Acceptable values are: **default**, **best-effort**, **restricted**, **single-numa-node**.

7.8.3. Pod interactions with Topology Manager policies

The example **Pod** specs below help illustrate pod interactions with Topology Manager.

The following pod runs in the **BestEffort** QoS class because no resource requests or limits are specified.

```
spec:
  containers:
  - name: nginx
    image: nginx
```

The next pod runs in the **Burstable** QoS class because requests are less than limits.

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
      requests:
        memory: "100Mi"
```

If the selected policy is anything other than **none**, Topology Manager would not consider either of these **Pod** specifications.

The last example pod below runs in the Guaranteed QoS class because requests are equal to limits.

```
spec:
  containers:
  - name: nginx
```

```
    image: nginx
    resources:
     limits:
       memory: "200Mi"
       cpu: "2"
       example.com/device: "1"
     requests:
       memory: "200Mi"
       cpu: "2"
       example.com/device: "1"
```

Topology Manager would consider this pod. The Topology Manager would consult the hint providers, which are CPU Manager and Device Manager, to get topology hints for the pod.

Topology Manager will use this information to store the best topology for this container. In the case of this pod, CPU Manager and Device Manager will use this stored information at the resource allocation stage.

7.9. RESOURCE REQUESTS AND OVERCOMMITMENT

For each compute resource, a container may specify a resource request and limit. Scheduling decisions are made based on the request to ensure that a node has enough capacity available to meet the requested value. If a container specifies limits, but omits requests, the requests are defaulted to the limits. A container is not able to exceed the specified limit on the node.

The enforcement of limits is dependent upon the compute resource type. If a container makes no request or limit, the container is scheduled to a node with no resource guarantees. In practice, the container is able to consume as much of the specified resource as is available with the lowest local priority. In low resource situations, containers that specify no resource requests are given the lowest quality of service.

Scheduling is based on resources requested, while quota and hard limits refer to resource limits, which can be set higher than requested resources. The difference between request and limit determines the level of overcommit; for instance, if a container is given a memory request of 1Gi and a memory limit of 2Gi, it is scheduled based on the 1Gi request being available on the node, but could use up to 2Gi; so it is 200% overcommitted.

7.10. CLUSTER-LEVEL OVERCOMMIT USING THE CLUSTER RESOURCE OVERRIDE OPERATOR

The Cluster Resource Override Operator is an admission webhook that allows you to control the level of overcommit and manage container density across all the nodes in your cluster. The Operator controls how nodes in specific projects can exceed defined memory and CPU limits.

You must install the Cluster Resource Override Operator using the OpenShift Container Platform console or CLI as shown in the following sections. During the installation, you create a **ClusterResourceOverride** custom resource (CR), where you set the level of overcommit, as shown in the following example:

```
    apiVersion: operator.autoscaling.openshift.io/v1
    kind: ClusterResourceOverride
    metadata:
      name: cluster 1
    spec:
```

```
podResourceOverride:
  spec:
    memoryRequestToLimitPercent: 50
    cpuRequestToLimitPercent: 25
    limitCPUToMemoryPercent: 200
# ...
```
2
3
4

1 The name must be **cluster**.

2 Optional. If a container memory limit has been specified or defaulted, the memory request is overridden to this percentage of the limit, between 1-100. The default is 50.

3 Optional. If a container CPU limit has been specified or defaulted, the CPU request is overridden to this percentage of the limit, between 1-100. The default is 25.

4 Optional. If a container memory limit has been specified or defaulted, the CPU limit is overridden to a percentage of the memory limit, if specified. Scaling 1Gi of RAM at 100 percent is equal to 1 CPU core. This is processed prior to overriding the CPU request (if configured). The default is 200.

> **NOTE**
>
> The Cluster Resource Override Operator overrides have no effect if limits have not been set on containers. Create a **LimitRange** object with default limits per individual project or configure limits in **Pod** specs for the overrides to apply.

When configured, overrides can be enabled per-project by applying the following label to the Namespace object for each project:

```
apiVersion: v1
kind: Namespace
metadata:

# ...

  labels:
    clusterresourceoverrides.admission.autoscaling.openshift.io/enabled: "true"

# ...
```

The Operator watches for the **ClusterResourceOverride** CR and ensures that the **ClusterResourceOverride** admission webhook is installed into the same namespace as the operator.

7.10.1. Installing the Cluster Resource Override Operator using the web console

You can use the OpenShift Container Platform web console to install the Cluster Resource Override Operator to help control overcommit in your cluster.

Prerequisites

- The Cluster Resource Override Operator has no effect if limits have not been set on containers. You must specify default limits for a project using a **LimitRange** object or configure limits in **Pod** specs for the overrides to apply.

Procedure

To install the Cluster Resource Override Operator using the OpenShift Container Platform web console:

1. In the OpenShift Container Platform web console, navigate to **Home → Projects**

   a. Click **Create Project**.

   b. Specify **clusterresourceoverride-operator** as the name of the project.

   c. Click **Create**.

2. Navigate to **Operators → OperatorHub**.

   a. Choose **ClusterResourceOverride Operator** from the list of available Operators and click **Install**.

   b. On the **Install Operator** page, make sure **A specific Namespace on the cluster** is selected for **Installation Mode**.

   c. Make sure **clusterresourceoverride-operator** is selected for **Installed Namespace**.

   d. Select an **Update Channel** and **Approval Strategy**.

   e. Click **Install**.

3. On the **Installed Operators** page, click **ClusterResourceOverride**.

   a. On the **ClusterResourceOverride Operator** details page, click **Create ClusterResourceOverride**.

   b. On the **Create ClusterResourceOverride** page, click **YAML view** and edit the YAML template to set the overcommit values as needed:

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster 1
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 2
      cpuRequestToLimitPercent: 25 3
      limitCPUToMemoryPercent: 200 4
# ...
```

1   The name must be **cluster**.

2   Optional. Specify the percentage to override the container memory limit, if used, between 1-100. The default is 50.

3   Optional. Specify the percentage to override the container CPU limit, if used, between 1-100. The default is 25.

4   Optional. Specify the percentage to override the container memory limit, if used. Scaling 1Gi of RAM at 100 percent is equal to 1 CPU core. This is processed prior to overriding the CPU request, if configured. The default is 200.

    c. Click **Create**.

4. Check the current state of the admission webhook by checking the status of the cluster custom resource:

    a. On the **ClusterResourceOverride Operator** page, click **cluster**.

    b. On the **ClusterResourceOverride Details** page, click **YAML**. The **mutatingWebhookConfigurationRef** section appears when the webhook is called.

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride","met
adata":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":{"spec":
{"cpuRequestToLimitPercent":25,"limitCPUToMemoryPercent":200,"memoryRequestToLi
mitPercent":50}}}}
  creationTimestamp: "2019-12-18T22:35:02Z"
  generation: 1
  name: cluster
  resourceVersion: "127622"
  selfLink: /apis/operator.autoscaling.openshift.io/v1/clusterresourceoverrides/cluster
  uid: 978fc959-1717-4bd1-97d0-ae00ee111e8d
spec:
  podResourceOverride:
    spec:
      cpuRequestToLimitPercent: 25
      limitCPUToMemoryPercent: 200
      memoryRequestToLimitPercent: 50
status:

# ...

    mutatingWebhookConfigurationRef: ①
      apiVersion: admissionregistration.k8s.io/v1
      kind: MutatingWebhookConfiguration
      name: clusterresourceoverrides.admission.autoscaling.openshift.io
      resourceVersion: "127621"
      uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

# ...
```

    ① Reference to the **ClusterResourceOverride** admission webhook.

7.10.2. Installing the Cluster Resource Override Operator using the CLI

You can use the OpenShift Container Platform CLI to install the Cluster Resource Override Operator to help control overcommit in your cluster.

Prerequisites

- The Cluster Resource Override Operator has no effect if limits have not been set on containers. You must specify default limits for a project using a **LimitRange** object or configure limits in **Pod** specs for the overrides to apply.

Procedure

To install the Cluster Resource Override Operator using the CLI:

1. Create a namespace for the Cluster Resource Override Operator:

   a. Create a **Namespace** object YAML file (for example, **cro-namespace.yaml**) for the Cluster Resource Override Operator:

   ```
   apiVersion: v1
   kind: Namespace
   metadata:
     name: clusterresourceoverride-operator
   ```

   b. Create the namespace:

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

   For example:

   ```
   $ oc create -f cro-namespace.yaml
   ```

2. Create an Operator group:

   a. Create an **OperatorGroup** object YAML file (for example, cro-og.yaml) for the Cluster Resource Override Operator:

   ```
   apiVersion: operators.coreos.com/v1
   kind: OperatorGroup
   metadata:
     name: clusterresourceoverride-operator
     namespace: clusterresourceoverride-operator
   spec:
     targetNamespaces:
       - clusterresourceoverride-operator
   ```

   b. Create the Operator Group:

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

   For example:

   ```
   $ oc create -f cro-og.yaml
   ```

3. Create a subscription:

   a. Create a **Subscription** object YAML file (for example, cro-sub.yaml) for the Cluster Resource Override Operator:

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   ```

```
metadata:
  name: clusterresourceoverride
  namespace: clusterresourceoverride-operator
spec:
  channel: "4.12"
  name: clusterresourceoverride
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

b. Create the subscription:

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

For example:

```
$ oc create -f cro-sub.yaml
```

4. Create a **ClusterResourceOverride** custom resource (CR) object in the **clusterresourceoverride-operator** namespace:

a. Change to the **clusterresourceoverride-operator** namespace.

```
$ oc project clusterresourceoverride-operator
```

b. Create a **ClusterResourceOverride** object YAML file (for example, cro-cr.yaml) for the Cluster Resource Override Operator:

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster 1
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 2
      cpuRequestToLimitPercent: 25 3
      limitCPUToMemoryPercent: 200 4
```

1 The name must be **cluster**.

2 Optional. Specify the percentage to override the container memory limit, if used, between 1-100. The default is 50.

3 Optional. Specify the percentage to override the container CPU limit, if used, between 1-100. The default is 25.

4 Optional. Specify the percentage to override the container memory limit, if used. Scaling 1Gi of RAM at 100 percent is equal to 1 CPU core. This is processed prior to overriding the CPU request, if configured. The default is 200.

c. Create the **ClusterResourceOverride** object:

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

For example:

```
$ oc create -f cro-cr.yaml
```

5. Verify the current state of the admission webhook by checking the status of the cluster custom resource.

```
$ oc get clusterresourceoverride cluster -n clusterresourceoverride-operator -o yaml
```

The **mutatingWebhookConfigurationRef** section appears when the webhook is called.

Example output

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride","metadata":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":{"spec":{"cpuRequestToLimitPercent":25,"limitCPUToMemoryPercent":200,"memoryRequestToLimitPercent":50}}}}
  creationTimestamp: "2019-12-18T22:35:02Z"
  generation: 1
  name: cluster
  resourceVersion: "127622"
  selfLink: /apis/operator.autoscaling.openshift.io/v1/clusterresourceoverrides/cluster
  uid: 978fc959-1717-4bd1-97d0-ae00ee111e8d
spec:
  podResourceOverride:
    spec:
      cpuRequestToLimitPercent: 25
      limitCPUToMemoryPercent: 200
      memoryRequestToLimitPercent: 50
status:

# ...

    mutatingWebhookConfigurationRef: ❶
      apiVersion: admissionregistration.k8s.io/v1
      kind: MutatingWebhookConfiguration
      name: clusterresourceoverrides.admission.autoscaling.openshift.io
      resourceVersion: "127621"
      uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

# ...
```

❶ Reference to the **ClusterResourceOverride** admission webhook.

7.10.3. Configuring cluster-level overcommit

The Cluster Resource Override Operator requires a **ClusterResourceOverride** custom resource (CR) and a label for each project where you want the Operator to control overcommit.

Prerequisites

- The Cluster Resource Override Operator has no effect if limits have not been set on containers. You must specify default limits for a project using a **LimitRange** object or configure limits in **Pod** specs for the overrides to apply.

Procedure

To modify cluster-level overcommit:

1. Edit the **ClusterResourceOverride** CR:

   ```
   apiVersion: operator.autoscaling.openshift.io/v1
   kind: ClusterResourceOverride
   metadata:
       name: cluster
   spec:
     podResourceOverride:
       spec:
         memoryRequestToLimitPercent: 50 ❶
         cpuRequestToLimitPercent: 25 ❷
         limitCPUToMemoryPercent: 200 ❸
   # ...
   ```

   ❶ Optional. Specify the percentage to override the container memory limit, if used, between 1-100. The default is 50.

   ❷ Optional. Specify the percentage to override the container CPU limit, if used, between 1-100. The default is 25.

   ❸ Optional. Specify the percentage to override the container memory limit, if used. Scaling 1Gi of RAM at 100 percent is equal to 1 CPU core. This is processed prior to overriding the CPU request, if configured. The default is 200.

2. Ensure the following label has been added to the Namespace object for each project where you want the Cluster Resource Override Operator to control overcommit:

   ```
   apiVersion: v1
   kind: Namespace
   metadata:

   # ...

     labels:
       clusterresourceoverrides.admission.autoscaling.openshift.io/enabled: "true" ❶

   # ...
   ```

   ❶ Add this label to each project.

7.11. NODE-LEVEL OVERCOMMIT

You can use various ways to control overcommit on specific nodes, such as quality of service (QOS) guarantees, CPU limits, or reserve resources. You can also disable overcommit for specific nodes and specific projects.

7.11.1. Understanding compute resources and containers

The node-enforced behavior for compute resources is specific to the resource type.

7.11.1.1. Understanding container CPU requests

A container is guaranteed the amount of CPU it requests and is additionally able to consume excess CPU available on the node, up to any limit specified by the container. If multiple containers are attempting to use excess CPU, CPU time is distributed based on the amount of CPU requested by each container.

For example, if one container requested 500m of CPU time and another container requested 250m of CPU time, then any extra CPU time available on the node is distributed among the containers in a 2:1 ratio. If a container specified a limit, it will be throttled not to use more CPU than the specified limit. CPU requests are enforced using the CFS shares support in the Linux kernel. By default, CPU limits are enforced using the CFS quota support in the Linux kernel over a 100ms measuring interval, though this can be disabled.

7.11.1.2. Understanding container memory requests

A container is guaranteed the amount of memory it requests. A container can use more memory than requested, but once it exceeds its requested amount, it could be terminated in a low memory situation on the node. If a container uses less memory than requested, it will not be terminated unless system tasks or daemons need more memory than was accounted for in the node's resource reservation. If a container specifies a limit on memory, it is immediately terminated if it exceeds the limit amount.

7.11.2. Understanding overcomitment and quality of service classes

A node is *overcommitted* when it has a pod scheduled that makes no request, or when the sum of limits across all pods on that node exceeds available machine capacity.

In an overcommitted environment, it is possible that the pods on the node will attempt to use more compute resource than is available at any given point in time. When this occurs, the node must give priority to one pod over another. The facility used to make this decision is referred to as a Quality of Service (QoS) Class.

A pod is designated as one of three QoS classes with decreasing order of priority:

Table 7.2. Quality of Service Classes

Priority	Class Name	Description
1 (highest)	**Guaranteed**	If limits and optionally requests are set (not equal to 0) for all resources and they are equal, then the pod is classified as **Guaranteed**.
2	**Burstable**	If requests and optionally limits are set (not equal to 0) for all resources, and they are not equal, then the pod is classified as **Burstable**.

Priority	Class Name	Description
3 (lowest)	**BestEffort**	If requests and limits are not set for any of the resources, then the pod is classified as **BestEffort**.

Memory is an incompressible resource, so in low memory situations, containers that have the lowest priority are terminated first:

- **Guaranteed** containers are considered top priority, and are guaranteed to only be terminated if they exceed their limits, or if the system is under memory pressure and there are no lower priority containers that can be evicted.

- **Burstable** containers under system memory pressure are more likely to be terminated once they exceed their requests and no other **BestEffort** containers exist.

- **BestEffort** containers are treated with the lowest priority. Processes in these containers are first to be terminated if the system runs out of memory.

7.11.2.1. Understanding how to reserve memory across quality of service tiers

You can use the **qos-reserved** parameter to specify a percentage of memory to be reserved by a pod in a particular QoS level. This feature attempts to reserve requested resources to exclude pods from lower OoS classes from using resources requested by pods in higher QoS classes.

OpenShift Container Platform uses the **qos-reserved** parameter as follows:

- A value of **qos-reserved=memory=100%** will prevent the **Burstable** and **BestEffort** QoS classes from consuming memory that was requested by a higher QoS class. This increases the risk of inducing OOM on **BestEffort** and **Burstable** workloads in favor of increasing memory resource guarantees for **Guaranteed** and **Burstable** workloads.

- A value of **qos-reserved=memory=50%** will allow the **Burstable** and **BestEffort** QoS classes to consume half of the memory requested by a higher QoS class.

- A value of **qos-reserved=memory=0%** will allow a **Burstable** and **BestEffort** QoS classes to consume up to the full node allocatable amount if available, but increases the risk that a **Guaranteed** workload will not have access to requested memory. This condition effectively disables this feature.

7.11.3. Understanding swap memory and QOS

You can disable swap by default on your nodes to preserve quality of service (QOS) guarantees. Otherwise, physical resources on a node can oversubscribe, affecting the resource guarantees the Kubernetes scheduler makes during pod placement.

For example, if two guaranteed pods have reached their memory limit, each container could start using swap memory. Eventually, if there is not enough swap space, processes in the pods can be terminated due to the system being oversubscribed.

Failing to disable swap results in nodes not recognizing that they are experiencing **MemoryPressure**, resulting in pods not receiving the memory they made in their scheduling request. As a result, additional pods are placed on the node to further increase memory pressure, ultimately increasing your risk of experiencing a system out of memory (OOM) event.
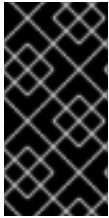
> **IMPORTANT**
>
> If swap is enabled, any out-of-resource handling eviction thresholds for available memory will not work as expected. Take advantage of out-of-resource handling to allow pods to be evicted from a node when it is under memory pressure, and rescheduled on an alternative node that has no such pressure.

7.11.4. Understanding nodes overcommitment

In an overcommitted environment, it is important to properly configure your node to provide best system behavior.

When the node starts, it ensures that the kernel tunable flags for memory management are set properly. The kernel should never fail memory allocations unless it runs out of physical memory.

To ensure this behavior, OpenShift Container Platform configures the kernel to always overcommit memory by setting the **vm.overcommit_memory** parameter to **1**, overriding the default operating system setting.

OpenShift Container Platform also configures the kernel not to panic when it runs out of memory by setting the **vm.panic_on_oom** parameter to **0**. A setting of 0 instructs the kernel to call oom_killer in an Out of Memory (OOM) condition, which kills processes based on priority

You can view the current setting by running the following commands on your nodes:

```
$ sysctl -a |grep commit
```

Example output

```
#...
vm.overcommit_memory = 0
#...
```

```
$ sysctl -a |grep panic
```

Example output

```
#...
vm.panic_on_oom = 0
#...
```

> **NOTE**
>
> The above flags should already be set on nodes, and no further action is required.

You can also perform the following configurations for each node:

- Disable or enforce CPU limits using CPU CFS quotas

- Reserve resources for system processes

- Reserve memory across quality of service tiers

7.11.5. Disabling or enforcing CPU limits using CPU CFS quotas

Nodes by default enforce specified CPU limits using the Completely Fair Scheduler (CFS) quota support in the Linux kernel.

If you disable CPU limit enforcement, it is important to understand the impact on your node:

- If a container has a CPU request, the request continues to be enforced by CFS shares in the Linux kernel.

- If a container does not have a CPU request, but does have a CPU limit, the CPU request defaults to the specified CPU limit, and is enforced by CFS shares in the Linux kernel.

- If a container has both a CPU request and limit, the CPU request is enforced by CFS shares in the Linux kernel, and the CPU limit has no impact on the node.

Prerequisites

- Obtain the label associated with the static **MachineConfigPool** CRD for the type of node you want to configure by entering the following command:

```
$ oc edit machineconfigpool <name>
```

For example:

```
$ oc edit machineconfigpool worker
```

Example output

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: ""  1
  name: worker
```

1 The label appears under Labels.

TIP

If the label is not present, add a key/value pair such as:

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

Procedure

1. Create a custom resource (CR) for your configuration change.

   Sample configuration for a disabling CPU limits

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: disable-cpu-units ❶
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" ❷
  kubeletConfig:
    cpuCfsQuota: false ❸
```

❶	Assign a name to CR.
❷	Specify the label from the machine config pool.
❸	Set the **cpuCfsQuota** parameter to **false**.

2. Run the following command to create the CR:

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

7.11.6. Reserving resources for system processes

To provide more reliable scheduling and minimize node resource overcommitment, each node can reserve a portion of its resources for use by system daemons that are required to run on your node for your cluster to function. In particular, it is recommended that you reserve resources for incompressible resources such as memory.

Procedure

To explicitly reserve resources for non-pod processes, allocate node resources by specifying resources available for scheduling. For more details, see Allocating Resources for Nodes.

7.11.7. Disabling overcommitment for a node

When enabled, overcommitment can be disabled on each node.

Procedure

To disable overcommitment in a node run the following command on that node:

```
$ sysctl -w vm.overcommit_memory=0
```

7.12. PROJECT-LEVEL LIMITS

To help control overcommit, you can set per-project resource limit ranges, specifying memory and CPU limits and defaults for a project that overcommit cannot exceed.

For information on project-level resource limits, see Additional resources.

Alternatively, you can disable overcommitment for specific projects.

7.12.1. Disabling overcommitment for a project

When enabled, overcommitment can be disabled per-project. For example, you can allow infrastructure components to be configured independently of overcommitment.

Procedure

To disable overcommitment in a project:

1. Edit the namespace object to add the following annotation:

   ```
   apiVersion: v1
   kind: Namespace
   metadata:
     annotations:
       quota.openshift.io/cluster-resource-override-enabled: "false" 1
   # ...
   ```

   1 Setting this annotation to **false** disables overcommit for this namespace.

7.13. FREEING NODE RESOURCES USING GARBAGE COLLECTION

Understand and use garbage collection.

7.13.1. Understanding how terminated containers are removed through garbage collection

Container garbage collection removes terminated containers by using eviction thresholds.

When eviction thresholds are set for garbage collection, the node tries to keep any container for any pod accessible from the API. If the pod has been deleted, the containers will be as well. Containers are preserved as long the pod is not deleted and the eviction threshold is not reached. If the node is under disk pressure, it will remove containers and their logs will no longer be accessible using **oc logs**.

- **eviction-soft** – A soft eviction threshold pairs an eviction threshold with a required administrator-specified grace period.

- **eviction-hard** – A hard eviction threshold has no grace period, and if observed, OpenShift Container Platform takes immediate action.

The following table lists the eviction thresholds:

Table 7.3. Variables for configuring container garbage collection

Node condition	Eviction signal	Description
MemoryPressure	**memory.available**	The available memory on the node.

Node condition	Eviction signal	Description
DiskPressure	**nodefs.available** **nodefs.inodesFree** **imagefs.available** **imagefs.inodesFree**	The available disk space or inodes on the node root file system, **nodefs**, or image file system, **imagefs**.

NOTE

For **evictionHard** you must specify all of these parameters. If you do not specify all parameters, only the specified parameters are applied and the garbage collection will not function properly.

If a node is oscillating above and below a soft eviction threshold, but not exceeding its associated grace period, the corresponding node would constantly oscillate between **true** and **false**. As a consequence, the scheduler could make poor scheduling decisions.

To protect against this oscillation, use the **eviction-pressure-transition-period** flag to control how long OpenShift Container Platform must wait before transitioning out of a pressure condition. OpenShift Container Platform will not set an eviction threshold as being met for the specified pressure condition for the period specified before toggling the condition back to false.

7.13.2. Understanding how images are removed through garbage collection

Image garbage collection removes images that are not referenced by any running pods.

OpenShift Container Platform determines which images to remove from a node based on the disk usage that is reported by **cAdvisor**.

The policy for image garbage collection is based on two conditions:

- The percent of disk usage (expressed as an integer) which triggers image garbage collection. The default is **85**.

- The percent of disk usage (expressed as an integer) to which image garbage collection attempts to free. Default is **80**.

For image garbage collection, you can modify any of the following variables using a custom resource.

Table 7.4. Variables for configuring image garbage collection

Setting	Description
imageMinimumGCAge	The minimum age for an unused image before the image is removed by garbage collection. The default is **2m**.
imageGCHighThresholdPercent	The percent of disk usage, expressed as an integer, which triggers image garbage collection. The default is **85**.

Setting	Description
imageGCLowThresh oldPercent	The percent of disk usage, expressed as an integer, to which image garbage collection attempts to free. The default is **80**.

Two lists of images are retrieved in each garbage collector run:

1. A list of images currently running in at least one pod.

2. A list of images available on a host.

As new containers are run, new images appear. All images are marked with a time stamp. If the image is running (the first list above) or is newly detected (the second list above), it is marked with the current time. The remaining images are already marked from the previous spins. All images are then sorted by the time stamp.

Once the collection starts, the oldest images get deleted first until the stopping criterion is met.

7.13.3. Configuring garbage collection for containers and images

As an administrator, you can configure how OpenShift Container Platform performs garbage collection by creating a **kubeletConfig** object for each machine config pool.

> **NOTE**
>
> OpenShift Container Platform supports only one **kubeletConfig** object for each machine config pool.

You can configure any combination of the following:

- Soft eviction for containers

- Hard eviction for containers

- Eviction for images

Container garbage collection removes terminated containers. Image garbage collection removes images that are not referenced by any running pods.

Prerequisites

1. Obtain the label associated with the static **MachineConfigPool** CRD for the type of node you want to configure by entering the following command:

```
$ oc edit machineconfigpool <name>
```

For example:

```
$ oc edit machineconfigpool worker
```

Example output

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: ""
  name: worker
#...
```
1

1   The label appears under Labels.

TIP

If the label is not present, add a key/value pair such as:

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

Procedure

1. Create a custom resource (CR) for your configuration change.

   ### IMPORTANT

   If there is one file system, or if **/var/lib/kubelet** and **/var/lib/containers/** are in the same file system, the settings with the highest values trigger evictions, as those are met first. The file system triggers the eviction.

   Sample configuration for a container garbage collection CR:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: KubeletConfig
   metadata:
     name: worker-kubeconfig
   spec:
     machineConfigPoolSelector:
       matchLabels:
         pools.operator.machineconfiguration.openshift.io/worker: ""
     kubeletConfig:
       evictionSoft:
         memory.available: "500Mi"
         nodefs.available: "10%"
         nodefs.inodesFree: "5%"
         imagefs.available: "15%"
         imagefs.inodesFree: "10%"
       evictionSoftGracePeriod:
         memory.available: "1m30s"
         nodefs.available: "1m30s"
         nodefs.inodesFree: "1m30s"
         imagefs.available: "1m30s"
         imagefs.inodesFree: "1m30s"
   ```
   1 **2** **3** **4** **5**

```
    evictionHard: 6
      memory.available: "200Mi"
      nodefs.available: "5%"
      nodefs.inodesFree: "4%"
      imagefs.available: "10%"
      imagefs.inodesFree: "5%"
    evictionPressureTransitionPeriod: 0s 7
    imageMinimumGCAge: 5m 8
    imageGCHighThresholdPercent: 80 9
    imageGCLowThresholdPercent: 75 10
  #...
```

1 Name for the object.

2 Specify the label from the machine config pool.

3 For container garbage collection: Type of eviction: **evictionSoft** or **evictionHard**.

4 For container garbage collection: Eviction thresholds based on a specific eviction trigger signal.

5 For container garbage collection: Grace periods for the soft eviction. This parameter does not apply to **eviction-hard**.

6 For container garbage collection: Eviction thresholds based on a specific eviction trigger signal. For **evictionHard** you must specify all of these parameters. If you do not specify all parameters, only the specified parameters are applied and the garbage collection will not function properly.

7 For container garbage collection: The duration to wait before transitioning out of an eviction pressure condition.

8 For image garbage collection: The minimum age for an unused image before the image is removed by garbage collection.

9 For image garbage collection: The percent of disk usage (expressed as an integer) that triggers image garbage collection.

10 For image garbage collection: The percent of disk usage (expressed as an integer) that image garbage collection attempts to free.

2. Run the following command to create the CR:

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

For example:

```
$ oc create -f gc-container.yaml
```

Example output

```
kubeletconfig.machineconfiguration.openshift.io/gc-container created
```

Verification

1. Verify that garbage collection is active by entering the following command. The Machine Config Pool you specified in the custom resource appears with **UPDATING** as 'true` until the change is fully implemented:

```
$ oc get machineconfigpool
```

Example output

```
NAME     CONFIG                               UPDATED   UPDATING
master   rendered-master-546383f80705bd5aeaba93   True      False
worker   rendered-worker-b4c51bb33ccaae6fc4a6a5   False     True
```

7.14. USING THE NODE TUNING OPERATOR

Understand and use the Node Tuning Operator.

The Node Tuning Operator helps you manage node-level tuning by orchestrating the TuneD daemon and achieves low latency performance by using the Performance Profile controller. The majority of high-performance applications require some level of kernel tuning. The Node Tuning Operator provides a unified management interface to users of node-level sysctls and more flexibility to add custom tuning specified by user needs.

The Operator manages the containerized TuneD daemon for OpenShift Container Platform as a Kubernetes daemon set. It ensures the custom tuning specification is passed to all containerized TuneD daemons running in the cluster in the format that the daemons understand. The daemons run on all nodes in the cluster, one per node.

Node-level settings applied by the containerized TuneD daemon are rolled back on an event that triggers a profile change or when the containerized TuneD daemon is terminated gracefully by receiving and handling a termination signal.

The Node Tuning Operator uses the Performance Profile controller to implement automatic tuning to achieve low latency performance for OpenShift Container Platform applications. The cluster administrator configures a performance profile to define node-level settings such as the following:

- Updating the kernel to kernel-rt.

- Choosing CPUs for housekeeping.

- Choosing CPUs for running workloads.

> **NOTE**
>
> Currently, disabling CPU load balancing is not supported by cgroup v2. As a result, you might not get the desired behavior from performance profiles if you have cgroup v2 enabled. Enabling cgroup v2 is not recommended if you are using performance profiles.

The Node Tuning Operator is part of a standard OpenShift Container Platform installation in version 4.1 and later.
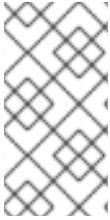
NOTE

In earlier versions of OpenShift Container Platform, the Performance Addon Operator was used to implement automatic tuning to achieve low latency performance for OpenShift applications. In OpenShift Container Platform 4.11 and later, this functionality is part of the Node Tuning Operator.

7.14.1. Accessing an example Node Tuning Operator specification

Use this process to access an example Node Tuning Operator specification.

Procedure

- Run the following command to access an example Node Tuning Operator specification:

```
oc get tuned.tuned.openshift.io/default -o yaml -n openshift-cluster-node-tuning-operator
```

The default CR is meant for delivering standard node-level tuning for the OpenShift Container Platform platform and it can only be modified to set the Operator Management state. Any other custom changes to the default CR will be overwritten by the Operator. For custom tuning, create your own Tuned CRs. Newly created CRs will be combined with the default CR and custom tuning applied to OpenShift Container Platform nodes based on node or pod labels and profile priorities.

WARNING

While in certain situations the support for pod labels can be a convenient way of automatically delivering required tuning, this practice is discouraged and strongly advised against, especially in large-scale clusters. The default Tuned CR ships without pod label matching. If a custom profile is created with pod label matching, then the functionality will be enabled at that time. The pod label functionality will be deprecated in future versions of the Node Tuning Operator.

7.14.2. Custom tuning specification

The custom resource (CR) for the Operator has two major sections. The first section, **profile:**, is a list of TuneD profiles and their names. The second, **recommend:**, defines the profile selection logic.

Multiple custom tuning specifications can co-exist as multiple CRs in the Operator's namespace. The existence of new CRs or the deletion of old CRs is detected by the Operator. All existing custom tuning specifications are merged and appropriate objects for the containerized TuneD daemons are updated.

Management state

The Operator Management state is set by adjusting the default Tuned CR. By default, the Operator is in the Managed state and the **spec.managementState** field is not present in the default Tuned CR. Valid values for the Operator Management state are as follows:

- Managed: the Operator will update its operands as configuration resources are updated

- Unmanaged: the Operator will ignore changes to the configuration resources

- Removed: the Operator will remove its operands and resources the Operator provisioned

Profile data

The **profile:** section lists TuneD profiles and their names.

```
profile:
- name: tuned_profile_1
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_1 profile

    [sysctl]
    net.ipv4.ip_forward=1
    # ... other sysctl's or other TuneD daemon plugins supported by the containerized TuneD

# ...

- name: tuned_profile_n
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_n profile

    # tuned_profile_n profile settings
```

Recommended profiles

The **profile:** selection logic is defined by the **recommend:** section of the CR. The **recommend:** section is a list of items to recommend the profiles based on a selection criteria.

```
recommend:
<recommend-item-1>
# ...
<recommend-item-n>
```

The individual items of the list:

```
- machineConfigLabels: ❶
    <mcLabels> ❷
  match: ❸
    <match> ❹
  priority: <priority> ❺
  profile: <tuned_profile_name> ❻
  operand: ❼
    debug: <bool> ❽
    tunedConfig:
      reapply_sysctl: <bool> ❾
```

❶ Optional.

❷ A dictionary of key/value **MachineConfig** labels. The keys must be unique.

3 If omitted, profile match is assumed unless a profile with a higher priority matches first or **machineConfigLabels** is set.

4 An optional list.

5 Profile ordering priority. Lower numbers mean higher priority (**0** is the highest priority).

6 A TuneD profile to apply on a match. For example **tuned_profile_1**.

7 Optional operand configuration.

8 Turn debugging on or off for the TuneD daemon. Options are **true** for on or **false** for off. The default is **false**.

9 Turn **reapply_sysctl** functionality on or off for the TuneD daemon. Options are **true** for on and **false** for off.

<match> is an optional list recursively defined as follows:

```
- label: <label_name>   ①
  value: <label_value>  ②
  type: <label_type>    ③
    <match>             ④
```

1 Node or pod label name.

2 Optional node or pod label value. If omitted, the presence of **<label_name>** is enough to match.

3 Optional object type (**node** or **pod**). If omitted, **node** is assumed.

4 An optional **<match>** list.

If **<match>** is not omitted, all nested **<match>** sections must also evaluate to **true**. Otherwise, **false** is assumed and the profile with the respective **<match>** section will not be applied or recommended. Therefore, the nesting (child **<match>** sections) works as logical AND operator. Conversely, if any item of the **<match>** list matches, the entire **<match>** list evaluates to **true**. Therefore, the list acts as logical OR operator.

If **machineConfigLabels** is defined, machine config pool based matching is turned on for the given **recommend:** list item. **<mcLabels>** specifies the labels for a machine config. The machine config is created automatically to apply host settings, such as kernel boot parameters, for the profile **<tuned_profile_name>**. This involves finding all machine config pools with machine config selector matching **<mcLabels>** and setting the profile **<tuned_profile_name>** on all nodes that are assigned the found machine config pools. To target nodes that have both master and worker roles, you must use the master role.

The list items **match** and **machineConfigLabels** are connected by the logical OR operator. The **match** item is evaluated first in a short-circuit manner. Therefore, if it evaluates to **true**, the **machineConfigLabels** item is not considered.
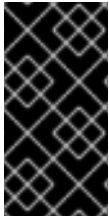
IMPORTANT

When using machine config pool based matching, it is advised to group nodes with the same hardware configuration into the same machine config pool. Not following this practice might result in TuneD operands calculating conflicting kernel parameters for two or more nodes sharing the same machine config pool.

Example: node or pod label based matching

```
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
    - label: node-role.kubernetes.io/master
    - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra
  priority: 20
  profile: openshift-control-plane
- priority: 30
  profile: openshift-node
```

The CR above is translated for the containerized TuneD daemon into its **recommend.conf** file based on the profile priorities. The profile with the highest priority (**10**) is **openshift-control-plane-es** and, therefore, it is considered first. The containerized TuneD daemon running on a given node looks to see if there is a pod running on the same node with the **tuned.openshift.io/elasticsearch** label set. If not, the entire **<match>** section evaluates as **false**. If there is such a pod with the label, in order for the **<match>** section to evaluate to **true**, the node label also needs to be **node-role.kubernetes.io/master** or **node-role.kubernetes.io/infra**.

If the labels for the profile with priority **10** matched, **openshift-control-plane-es** profile is applied and no other profile is considered. If the node/pod label combination did not match, the second highest priority profile (**openshift-control-plane**) is considered. This profile is applied if the containerized TuneD pod runs on a node with labels **node-role.kubernetes.io/master** or **node-role.kubernetes.io/infra**.

Finally, the profile **openshift-node** has the lowest priority of **30**. It lacks the **<match>** section and, therefore, will always match. It acts as a profile catch-all to set **openshift-node** profile, if no other profile with higher priority matches on a given node.
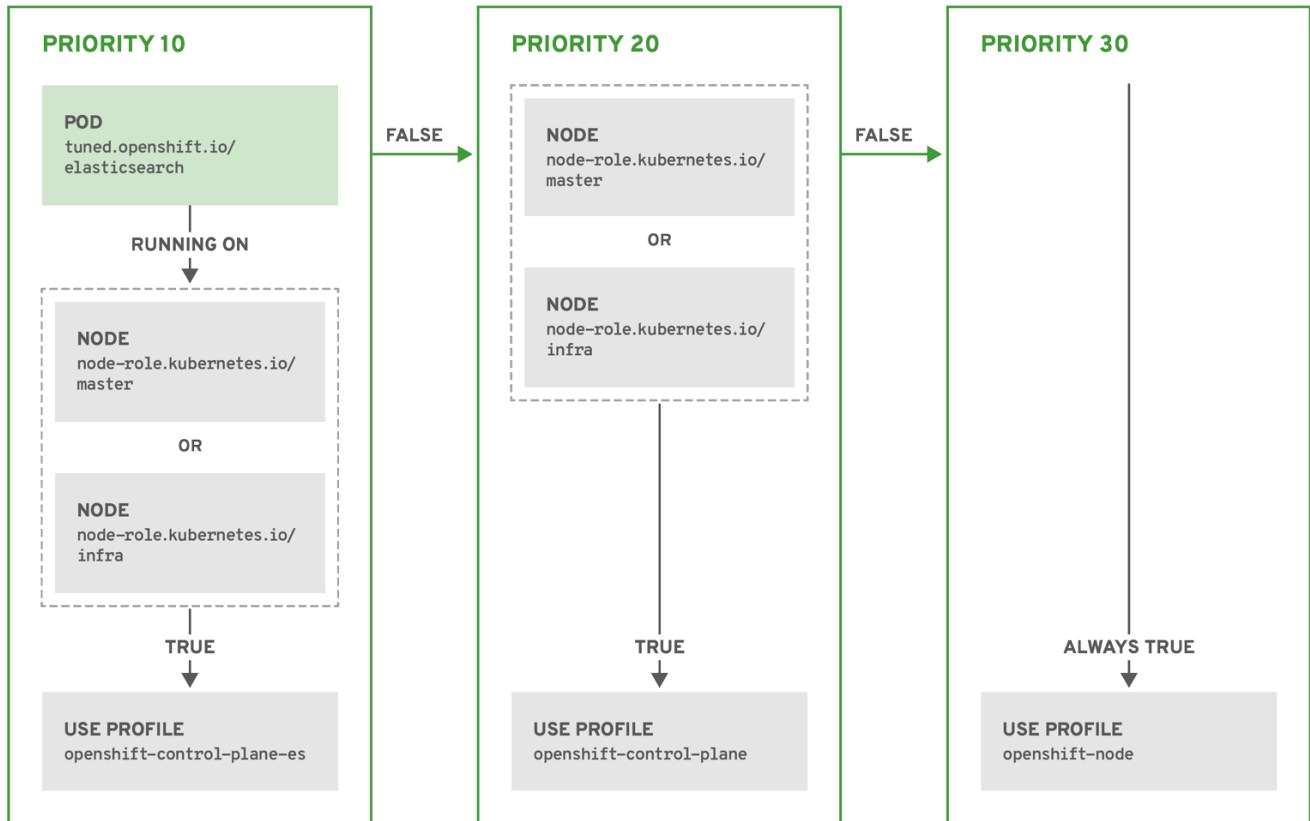
OPENSHIFT_10_0319

Example: machine config pool based matching

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: openshift-node-custom
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=Custom OpenShift node profile with an additional kernel parameter
      include=openshift-node
      [bootloader]
      cmdline_openshift_node_custom=+skew_tick=1
    name: openshift-node-custom

  recommend:
  - machineConfigLabels:
      machineconfiguration.openshift.io/role: "worker-custom"
    priority: 20
    profile: openshift-node-custom
```

To minimize node reboots, label the target nodes with a label the machine config pool's node selector will match, then create the Tuned CR above and finally create the custom machine config pool itself.

Cloud provider-specific TuneD profiles

With this functionality, all Cloud provider-specific nodes can conveniently be assigned a TuneD profile specifically tailored to a given Cloud provider on a OpenShift Container Platform cluster. This can be accomplished without adding additional node labels or grouping nodes into machine config pools.

This functionality takes advantage of **spec.providerID** node object values in the form of **<cloud-provider>://<cloud-provider-specific-id>** and writes the file **/var/lib/tuned/provider** with the value **<cloud-provider>** in NTO operand containers. The content of this file is then used by TuneD to load **provider-<cloud-provider>** profile if such profile exists.

The **openshift** profile that both **openshift-control-plane** and **openshift-node** profiles inherit settings from is now updated to use this functionality through the use of conditional profile loading. Neither NTO nor TuneD currently ship any Cloud provider-specific profiles. However, it is possible to create a custom profile **provider-<cloud-provider>** that will be applied to all Cloud provider-specific cluster nodes.

Example GCE Cloud provider profile

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: provider-gce
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=GCE Cloud provider-specific profile
      # Your tuning for GCE Cloud provider goes here.
    name: provider-gce
```

> **NOTE**
>
> Due to profile inheritance, any setting specified in the **provider-<cloud-provider>** profile will be overwritten by the **openshift** profile and its child profiles.

7.14.3. Default profiles set on a cluster

The following are the default profiles set on a cluster.

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: default
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=Optimize systems running OpenShift (provider specific parent profile)
      include=-provider-${f:exec:cat:/var/lib/tuned/provider},openshift
    name: openshift
  recommend:
  - profile: openshift-control-plane
    priority: 30
    match:
    - label: node-role.kubernetes.io/master
```

```
      - label: node-role.kubernetes.io/infra
    - profile: openshift-node
      priority: 40
```

Starting with OpenShift Container Platform 4.9, all OpenShift TuneD profiles are shipped with the TuneD package. You can use the **oc exec** command to view the contents of these profiles:

```
$ oc exec $tuned_pod -n openshift-cluster-node-tuning-operator -- find /usr/lib/tuned/openshift{,-control-plane,-node} -name tuned.conf -exec grep -H ^ {} \;
```

7.14.4. Supported TuneD daemon plugins

Excluding the **[main]** section, the following TuneD plugins are supported when using custom profiles defined in the **profile:** section of the Tuned CR:

- audio

- cpu

- disk

- eeepc_she

- modules

- mounts

- net

- scheduler

- scsi_host

- selinux

- sysctl

- sysfs

- usb

- video

- vm

- bootloader

There is some dynamic tuning functionality provided by some of these plugins that is not supported. The following TuneD plugins are currently not supported:

- script

- systemd

NOTE

The TuneD bootloader plugin only supports Red Hat Enterprise Linux CoreOS (RHCOS) worker nodes.

Additional resources

- [Available TuneD Plugins](#)

- [Getting Started with TuneD](#)

7.15. CONFIGURING THE MAXIMUM NUMBER OF PODS PER NODE

Two parameters control the maximum number of pods that can be scheduled to a node: **podsPerCore** and **maxPods**. If you use both options, the lower of the two limits the number of pods on a node.

For example, if **podsPerCore** is set to **10** on a node with 4 processor cores, the maximum number of pods allowed on the node will be 40.

Prerequisites

1. Obtain the label associated with the static **MachineConfigPool** CRD for the type of node you want to configure by entering the following command:

   ```
   $ oc edit machineconfigpool <name>
   ```

   For example:

   ```
   $ oc edit machineconfigpool worker
   ```

   Example output

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfigPool
   metadata:
     creationTimestamp: "2022-11-16T15:34:25Z"
     generation: 4
     labels:
       pools.operator.machineconfiguration.openshift.io/worker: ""  ❶
     name: worker
   #...
   ```

   ❶ The label appears under Labels.

   TIP

   If the label is not present, add a key/value pair such as:

   ```
   $ oc label machineconfigpool worker custom-kubelet=small-pods
   ```

Procedure

1. Create a custom resource (CR) for your configuration change.

   Sample configuration for a **max-pods** CR

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: KubeletConfig
   metadata:
     name: set-max-pods 1
   spec:
     machineConfigPoolSelector:
       matchLabels:
         pools.operator.machineconfiguration.openshift.io/worker: "" 2
     kubeletConfig:
       podsPerCore: 10 3
       maxPods: 250 4
   #...
   ```

   1   Assign a name to CR.

   2   Specify the label from the machine config pool.

   3   Specify the number of pods the node can run based on the number of processor cores on the node.

   4   Specify the number of pods the node can run to a fixed value, regardless of the properties of the node.

   > **NOTE**
   >
   > Setting **podsPerCore** to **0** disables this limit.

   In the above example, the default value for **podsPerCore** is **10** and the default value for **maxPods** is **250**. This means that unless the node has 25 cores or more, by default, **podsPerCore** will be the limiting factor.

2. Run the following command to create the CR:

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

Verification

1. List the **MachineConfigPool** CRDs to see if the change is applied. The **UPDATING** column reports **True** if the change is picked up by the Machine Config Controller:

   ```
   $ oc get machineconfigpools
   ```

   Example output

   ```
   NAME    CONFIG                   UPDATED  UPDATING  DEGRADED
   master  master-9cc2c72f205e103bb534  False    False     False
   worker  worker-8cecd1236b33ee3f8a5e  False    True      False
   ```

Once the change is complete, the **UPDATED** column reports **True**.

```
$ oc get machineconfigpools
```

Example output

```
NAME     CONFIG                    UPDATED   UPDATING   DEGRADED
master   master-9cc2c72f205e103bb534   False     True       False
worker   worker-8cecd1236b33ee3f8a5e   True      False      False
```

CHAPTER 8. POSTINSTALLATION NETWORK CONFIGURATION

After installing OpenShift Container Platform, you can further expand and customize your network to your requirements.

8.1. CLUSTER NETWORK OPERATOR CONFIGURATION

The configuration for the cluster network is specified as part of the Cluster Network Operator (CNO) configuration and stored in a custom resource (CR) object that is named **cluster**. The CR specifies the fields for the **Network** API in the **operator.openshift.io** API group.

The CNO configuration inherits the following fields during cluster installation from the **Network** API in the **Network.config.openshift.io** API group and these fields cannot be changed:

clusterNetwork

IP address pools from which pod IP addresses are allocated.

serviceNetwork

IP address pool for services.

defaultNetwork.type

Cluster network plugin, such as OpenShift SDN or OVN-Kubernetes.

> **NOTE**
>
> After cluster installation, you cannot modify the fields listed in the previous section.

8.2. ENABLING THE CLUSTER-WIDE PROXY

The **Proxy** object is used to manage the cluster-wide egress proxy. When a cluster is installed or upgraded without the proxy configured, a **Proxy** object is still generated but it will have a nil **spec**. For example:

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  trustedCA:
    name: ""
status:
```

A cluster administrator can configure the proxy for OpenShift Container Platform by modifying this **cluster Proxy** object.

> **NOTE**
>
> Only the **Proxy** object named **cluster** is supported, and no additional proxies can be created.

Prerequisites

- Cluster administrator permissions

- OpenShift Container Platform **oc** CLI tool installed

Procedure

1. Create a config map that contains any additional CA certificates required for proxying HTTPS connections.

   > **NOTE**
   >
   > You can skip this step if the proxy's identity certificate is signed by an authority from the RHCOS trust bundle.

   a. Create a file called **user-ca-bundle.yaml** with the following contents, and provide the values of your PEM-encoded certificates:

   ```
   apiVersion: v1
   data:
     ca-bundle.crt: | ❶
       <MY_PEM_ENCODED_CERTS> ❷
   kind: ConfigMap
   metadata:
     name: user-ca-bundle ❸
     namespace: openshift-config ❹
   ```

   ❶ This data key must be named **ca-bundle.crt**.

   ❷ One or more PEM-encoded X.509 certificates used to sign the proxy's identity certificate.

   ❸ The config map name that will be referenced from the **Proxy** object.

   ❹ The config map must be in the **openshift-config** namespace.

   b. Create the config map from this file:

   ```
   $ oc create -f user-ca-bundle.yaml
   ```

2. Use the **oc edit** command to modify the **Proxy** object:

   ```
   $ oc edit proxy/cluster
   ```

3. Configure the necessary fields for the proxy:

   ```
   apiVersion: config.openshift.io/v1
   kind: Proxy
   metadata:
     name: cluster
   spec:
     httpProxy: http://<username>:<pswd>@<ip>:<port> ❶
     httpsProxy: https://<username>:<pswd>@<ip>:<port> ❷
   ```

```
    noProxy: example.com  3
    readinessEndpoints:
    - http://www.google.com  4
    - https://www.google.com
    trustedCA:
      name: user-ca-bundle  5
```

1 A proxy URL to use for creating HTTP connections outside the cluster. The URL scheme must be **http**.

2 A proxy URL to use for creating HTTPS connections outside the cluster. The URL scheme must be either **http** or **https**. Specify a URL for the proxy that supports the URL scheme. For example, most proxies will report an error if they are configured to use **https** but they only support **http**. This failure message may not propagate to the logs and can appear to be a network connection failure instead. If using a proxy that listens for **https** connections from the cluster, you may need to configure the cluster to accept the CAs and certificates that the proxy uses.

3 A comma-separated list of destination domain names, domains, IP addresses or other network CIDRs to exclude proxying.

Preface a domain with **.** to match subdomains only. For example, **.y.com** matches **x.y.com**, but not **y.com**. Use ***** to bypass proxy for all destinations. If you scale up workers that are not included in the network defined by the **networking.machineNetwork[].cidr** field from the installation configuration, you must add them to this list to prevent connection issues.

This field is ignored if neither the **httpProxy** or **httpsProxy** fields are set.

4 One or more URLs external to the cluster to use to perform a readiness check before writing the **httpProxy** and **httpsProxy** values to status.

5 A reference to the config map in the **openshift-config** namespace that contains additional CA certificates required for proxying HTTPS connections. Note that the config map must already exist before referencing it here. This field is required unless the proxy's identity certificate is signed by an authority from the RHCOS trust bundle.

4. Save the file to apply the changes.

8.3. SETTING DNS TO PRIVATE

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

Procedure

1. Review the **DNS** custom resource for your cluster:

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

   Example output

   ```
   apiVersion: config.openshift.io/v1
   kind: DNS
   metadata:
     creationTimestamp: "2019-10-25T18:27:09Z"
   ```

```
  generation: 2
  name: cluster
  resourceVersion: "37966"
  selfLink: /apis/config.openshift.io/v1/dnses/cluster
  uid: 0e714746-f755-11f9-9cb1-02ff55d8f976
spec:
  baseDomain: <base_domain>
  privateZone:
    tags:
      Name: <infrastructure_id>-int
      kubernetes.io/cluster/<infrastructure_id>: owned
  publicZone:
    id: Z2XXXXXXXXXXA4
status: {}
```

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

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

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

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

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

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

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

Example output

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

8.4. CONFIGURING INGRESS CLUSTER TRAFFIC

OpenShift Container Platform provides the following methods for communicating from outside the cluster with services running in the cluster:

- If you have HTTP/HTTPS, use an Ingress Controller.

- If you have a TLS-encrypted protocol other than HTTPS, such as TLS with the SNI header, use an Ingress Controller.

- Otherwise, use a load balancer, an external IP, or a node port.

Method	Purpose
Use an Ingress Controller	Allows access to HTTP/HTTPS traffic and TLS-encrypted protocols other than HTTPS, such as TLS with the SNI header.
Automatically assign an external IP by using a load balancer service	Allows traffic to non-standard ports through an IP address assigned from a pool.
Manually assign an external IP to a service	Allows traffic to non-standard ports through a specific IP address.
Configure a **NodePort**	Expose a service on all nodes in the cluster.

8.5. CONFIGURING THE NODE PORT SERVICE RANGE

As a cluster administrator, you can expand the available node port range. If your cluster uses of a large number of node ports, you might need to increase the number of available ports.

The default port range is **30000-32767**. You can never reduce the port range, even if you first expand it beyond the default range.

8.5.1. Prerequisites

- Your cluster infrastructure must allow access to the ports that you specify within the expanded range. For example, if you expand the node port range to **30000-32900**, the inclusive port range of **32768-32900** must be allowed by your firewall or packet filtering configuration.

8.5.1.1. Expanding the node port range

You can expand the node port range for the cluster.

Prerequisites

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

- Log in to the cluster with a user with **cluster-admin** privileges.

Procedure

1. To expand the node port range, enter the following command. Replace **<port>** with the largest port number in the new range.

```
$ oc patch network.config.openshift.io cluster --type=merge -p \
 '{
   "spec":
     { "serviceNodePortRange": "30000-<port>" }
 }'
```

TIP

You can alternatively apply the following YAML to update the node port range:

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  serviceNodePortRange: "30000-<port>"
```

Example output

```
network.config.openshift.io/cluster patched
```

2. To confirm that the configuration is active, enter the following command. It can take several minutes for the update to apply.

```
$ oc get configmaps -n openshift-kube-apiserver config \
  -o jsonpath="{.data['config\.yaml']}" | \
  grep -Eo '"service-node-port-range":["[[:digit:]]+-[[:digit:]]+"]'
```

Example output

```
"service-node-port-range":["30000-33000"]
```

8.6. CONFIGURING IPSEC ENCRYPTION

With IPsec enabled, all network traffic between nodes on the OVN-Kubernetes network plugin travels through an encrypted tunnel.

IPsec is disabled by default.

8.6.1. Prerequisites

- Your cluster must use the OVN-Kubernetes network plugin.

8.6.1.1. Enabling IPsec encryption

As a cluster administrator, you can enable IPsec encryption after cluster installation.

Prerequisites

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

- Log in to the cluster with a user with **cluster-admin** privileges.

- You have reduced the size of your cluster MTU by **46** bytes to allow for the overhead of the IPsec ESP header.

Procedure

- To enable IPsec encryption, enter the following command:

```
$ oc patch networks.operator.openshift.io cluster --type=merge \
-p '{"spec":{"defaultNetwork":{"ovnKubernetesConfig":{"ipsecConfig":{ }}}}}'
```

Verification

1. To find the names of the OVN-Kubernetes control plane pods, enter the following command:

```
$ oc get pods -l app=ovnkube-master -n openshift-ovn-kubernetes
```

Example output

```
NAME                READY   STATUS    RESTARTS   AGE
ovnkube-master-fvtnh   6/6    Running   0         122m
ovnkube-master-hsgmm   6/6    Running   0         122m
ovnkube-master-qcmdc   6/6    Running   0         122m
```

2. Verify that IPsec is enabled on your cluste by running the following command:

```
$ oc -n openshift-ovn-kubernetes rsh ovnkube-master-<XXXXX> \
  ovn-nbctl --no-leader-only get nb_global . ipsec
```

where:

<XXXXX>

Specifies the random sequence of letters for a pod from the previous step.

Example output

```
true
```

8.7. CONFIGURING NETWORK POLICY

As a cluster administrator or project administrator, you can configure network policies for a project.

8.7.1. About network policy

In a cluster using a network plugin that supports Kubernetes network policy, network isolation is controlled entirely by **NetworkPolicy** objects. In OpenShift Container Platform 4.12, OpenShift SDN supports using network policy in its default network isolation mode.

> **WARNING**
>
> Network policy does not apply to the host network namespace. Pods with host networking enabled are unaffected by network policy rules. However, pods connecting to the host-networked pods might be affected by the network policy rules.
>
> Network policies cannot block traffic from localhost or from their resident nodes.

By default, all pods in a project are accessible from other pods and network endpoints. To isolate one or more pods in a project, you can create **NetworkPolicy** objects in that project to indicate the allowed incoming connections. Project administrators can create and delete **NetworkPolicy** objects within their own project.

If a pod is matched by selectors in one or more **NetworkPolicy** objects, then the pod will accept only connections that are allowed by at least one of those **NetworkPolicy** objects. A pod that is not selected by any **NetworkPolicy** objects is fully accessible.

A network policy applies to only the TCP, UDP, ICMP, and SCTP protocols. Other protocols are not affected.

The following example **NetworkPolicy** objects demonstrate supporting different scenarios:

- Deny all traffic:
  To make a project deny by default, add a **NetworkPolicy** object that matches all pods but accepts no traffic:

  ```
  kind: NetworkPolicy
  apiVersion: networking.k8s.io/v1
  metadata:
    name: deny-by-default
  spec:
    podSelector: {}
    ingress: []
  ```

- Only allow connections from the OpenShift Container Platform Ingress Controller:
  To make a project allow only connections from the OpenShift Container Platform Ingress Controller, add the following **NetworkPolicy** object.

  ```
  apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-openshift-ingress
  spec:
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            network.openshift.io/policy-group: ingress
  ```

```
    podSelector: {}
    policyTypes:
    - Ingress
```

- Only accept connections from pods within a project:
  To make pods accept connections from other pods in the same project, but reject all other connections from pods in other projects, add the following **NetworkPolicy** object:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector: {}
  ingress:
  - from:
    - podSelector: {}
```

- Only allow HTTP and HTTPS traffic based on pod labels:
  To enable only HTTP and HTTPS access to the pods with a specific label (**role=frontend** in following example), add a **NetworkPolicy** object similar to the following:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-http-and-https
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
  - ports:
    - protocol: TCP
      port: 80
    - protocol: TCP
      port: 443
```

- Accept connections by using both namespace and pod selectors:
  To match network traffic by combining namespace and pod selectors, you can use a **NetworkPolicy** object similar to the following:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-pod-and-namespace-both
spec:
  podSelector:
    matchLabels:
      name: test-pods
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          project: project_name
```

```
      podSelector:
        matchLabels:
          name: test-pods
```

NetworkPolicy objects are additive, which means you can combine multiple **NetworkPolicy** objects together to satisfy complex network requirements.

For example, for the **NetworkPolicy** objects defined in previous samples, you can define both **allow-same-namespace** and **allow-http-and-https** policies within the same project. Thus allowing the pods with the label **role=frontend**, to accept any connection allowed by each policy. That is, connections on any port from pods in the same namespace, and connections on ports **80** and **443** from pods in any namespace.

8.7.1.1. Using the allow-from-router network policy

Use the following **NetworkPolicy** to allow external traffic regardless of the router configuration:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-router
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/ingress: "" 1
  podSelector: {}
  policyTypes:
  - Ingress
```

1   **policy-group.network.openshift.io/ingress:""** label supports both OpenShift-SDN and OVN-Kubernetes.

8.7.1.2. Using the allow-from-hostnetwork network policy

Add the following **allow-from-hostnetwork NetworkPolicy** object to direct traffic from the host network pods:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-hostnetwork
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/host-network: ""
  podSelector: {}
  policyTypes:
  - Ingress
```

8.7.2. Example NetworkPolicy object

The following annotates an example NetworkPolicy object:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 1
spec:
  podSelector: 2
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: 3
        matchLabels:
          app: app
    ports: 4
    - protocol: TCP
      port: 27017
```

1   The name of the NetworkPolicy object.

2   A selector that describes the pods to which the policy applies. The policy object can only select pods in the project that defines the NetworkPolicy object.

3   A selector that matches the pods from which the policy object allows ingress traffic. The selector matches pods in the same namespace as the NetworkPolicy.

4   A list of one or more destination ports on which to accept traffic.

8.7.3. Creating a network policy using the CLI

To define granular rules describing ingress or egress network traffic allowed for namespaces in your cluster, you can create a network policy.

> **NOTE**
>
> If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OpenShift SDN network provider with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

- You installed the OpenShift CLI (**oc**).

- You are logged in to the cluster with a user with **admin** privileges.

- You are working in the namespace that the network policy applies to.

Procedure

1. Create a policy rule:

    a. Create a **<policy_name>.yaml** file:

    ```
    $ touch <policy_name>.yaml
    ```

    where:

    <policy_name>

    Specifies the network policy file name.

    b. Define a network policy in the file that you just created, such as in the following examples:

    Deny ingress from all pods in all namespaces

    This is a fundamental policy, blocking all cross-pod networking other than cross-pod traffic allowed by the configuration of other Network Policies.

    ```
    kind: NetworkPolicy
    apiVersion: networking.k8s.io/v1
    metadata:
      name: deny-by-default
    spec:
      podSelector:
      ingress: []
    ```

    Allow ingress from all pods in the same namespace

    ```
    kind: NetworkPolicy
    apiVersion: networking.k8s.io/v1
    metadata:
      name: allow-same-namespace
    spec:
      podSelector:
      ingress:
      - from:
        - podSelector: {}
    ```

    Allow ingress traffic to one pod from a particular namespace

    This policy allows traffic to pods labelled **pod-a** from pods running in **namespace-y**.

    ```
    kind: NetworkPolicy
    apiVersion: networking.k8s.io/v1
    metadata:
      name: allow-traffic-pod
    spec:
      podSelector:
       matchLabels:
          pod: pod-a
      policyTypes:
      - Ingress
    ```

```
      ingress:
      - from:
        - namespaceSelector:
            matchLabels:
                kubernetes.io/metadata.name: namespace-y
```

2. To create the network policy object, enter the following command:

   ```
   $ oc apply -f <policy_name>.yaml -n <namespace>
   ```

   where:

   <policy_name>

   Specifies the network policy file name.

   <namespace>

   Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

   Example output

   ```
   networkpolicy.networking.k8s.io/deny-by-default created
   ```

> **NOTE**
>
> If you log in to the web console with **cluster-admin** privileges, you have a choice of creating a network policy in any namespace in the cluster directly in YAML or from a form in the web console.

8.7.4. Configuring multitenant isolation by using network policy

You can configure your project to isolate it from pods and services in other project namespaces.

Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OpenShift SDN network provider with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

- You installed the OpenShift CLI (**oc**).

- You are logged in to the cluster with a user with **admin** privileges.

Procedure

1. Create the following **NetworkPolicy** objects:

   a. A policy named **allow-from-openshift-ingress**.

      ```
      $ cat << EOF| oc create -f -
      apiVersion: networking.k8s.io/v1
      kind: NetworkPolicy
      metadata:
        name: allow-from-openshift-ingress
      ```

```
    spec:
      ingress:
      - from:
        - namespaceSelector:
            matchLabels:
              policy-group.network.openshift.io/ingress: ""
      podSelector: {}
      policyTypes:
      - Ingress
    EOF
```

> **NOTE**
>
> **policy-group.network.openshift.io/ingress: ""** is the preferred namespace
> selector label for OpenShift SDN. You can use the
> **network.openshift.io/policy-group: ingress** namespace selector label, but
> this is a legacy label.

b. A policy named **allow-from-openshift-monitoring**:

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-monitoring
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: monitoring
  podSelector: {}
  policyTypes:
  - Ingress
EOF
```

c. A policy named **allow-same-namespace**:

```
$ cat << EOF| oc create -f -
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
EOF
```

d. A policy named **allow-from-kube-apiserver-operator**:

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
```

```
kind: NetworkPolicy
metadata:
  name: allow-from-kube-apiserver-operator
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: openshift-kube-apiserver-operator
      podSelector:
        matchLabels:
          app: kube-apiserver-operator
  policyTypes:
  - Ingress
EOF
```

For more details, see New **kube-apiserver-operator** webhook controller validating health of webhook.

2. Optional: To confirm that the network policies exist in your current project, enter the following command:

```
$ oc describe networkpolicy
```

Example output

```
Name:         allow-from-openshift-ingress
Namespace:    example1
Created on:   2020-06-09 00:28:17 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      NamespaceSelector: network.openshift.io/policy-group: ingress
  Not affecting egress traffic
  Policy Types: Ingress


Name:         allow-from-openshift-monitoring
Namespace:    example1
Created on:   2020-06-09 00:29:57 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      NamespaceSelector: network.openshift.io/policy-group: monitoring
  Not affecting egress traffic
  Policy Types: Ingress
```

8.7.5. Creating default network policies for a new project

As a cluster administrator, you can modify the new project template to automatically include **NetworkPolicy** objects when you create a new project.

8.7.6. Modifying the template for new projects

As a cluster administrator, you can modify the default project template so that new projects are created using your custom requirements.

To create your own custom project template:

Procedure

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

2. Generate the default project template:

   ```
   $ oc adm create-bootstrap-project-template -o yaml > template.yaml
   ```

3. Use a text editor to modify the generated **template.yaml** file by adding objects or modifying existing objects.

4. The project template must be created in the **openshift-config** namespace. Load your modified template:

   ```
   $ oc create -f template.yaml -n openshift-config
   ```

5. Edit the project configuration resource using the web console or CLI.

   - Using the web console:

     i. Navigate to the **Administration** → **Cluster Settings** page.

     ii. Click **Configuration** to view all configuration resources.

     iii. Find the entry for **Project** and click **Edit YAML**.

   - Using the CLI:

     i. Edit the **project.config.openshift.io/cluster** resource:

        ```
        $ oc edit project.config.openshift.io/cluster
        ```

6. Update the **spec** section to include the **projectRequestTemplate** and **name** parameters, and set the name of your uploaded project template. The default name is **project-request**.

   Project configuration resource with custom project template

   ```
   apiVersion: config.openshift.io/v1
   kind: Project
   metadata:
   # ...
   spec:
   ```

```
    projectRequestTemplate:
      name: <template_name>
    # ...
```

7. After you save your changes, create a new project to verify that your changes were successfully applied.

8.7.6.1. Adding network policies to the new project template

As a cluster administrator, you can add network policies to the default template for new projects. OpenShift Container Platform will automatically create all the **NetworkPolicy** objects specified in the template in the project.

Prerequisites

- Your cluster uses a default CNI network provider that supports **NetworkPolicy** objects, such as the OpenShift SDN network provider with **mode: NetworkPolicy** set. This mode is the default for OpenShift SDN.

- You installed the OpenShift CLI (**oc**).

- You must log in to the cluster with a user with **cluster-admin** privileges.

- You must have created a custom default project template for new projects.

Procedure

1. Edit the default template for a new project by running the following command:

   ```
   $ oc edit template <project_template> -n openshift-config
   ```

   Replace **<project_template>** with the name of the default template that you configured for your cluster. The default template name is **project-request**.

2. In the template, add each **NetworkPolicy** object as an element to the **objects** parameter. The **objects** parameter accepts a collection of one or more objects.
   In the following example, the **objects** parameter collection includes several **NetworkPolicy** objects.

   ```
   objects:
   - apiVersion: networking.k8s.io/v1
     kind: NetworkPolicy
     metadata:
       name: allow-from-same-namespace
     spec:
       podSelector: {}
       ingress:
       - from:
         - podSelector: {}
   - apiVersion: networking.k8s.io/v1
     kind: NetworkPolicy
     metadata:
       name: allow-from-openshift-ingress
     spec:
       ingress:
   ```

```
      - from:
       - namespaceSelector:
          matchLabels:
            network.openshift.io/policy-group: ingress
     podSelector: {}
     policyTypes:
     - Ingress
  - apiVersion: networking.k8s.io/v1
    kind: NetworkPolicy
    metadata:
     name: allow-from-kube-apiserver-operator
    spec:
     ingress:
     - from:
       - namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: openshift-kube-apiserver-operator
        podSelector:
          matchLabels:
            app: kube-apiserver-operator
     policyTypes:
     - Ingress
  ...
```

3. Optional: Create a new project to confirm that your network policy objects are created successfully by running the following commands:

   a. Create a new project:

   ```
   $ oc new-project <project>  ❶
   ```

   ❶   Replace **<project>** with the name for the project you are creating.

   b. Confirm that the network policy objects in the new project template exist in the new project:

   ```
   $ oc get networkpolicy
   NAME                       POD-SELECTOR   AGE
   allow-from-openshift-ingress   <none>       7s
   allow-from-same-namespace      <none>       7s
   ```

8.8. OPTIMIZING ROUTING

The OpenShift Container Platform HAProxy router can be scaled or configured to optimize performance.

8.8.1. Baseline Ingress Controller (router) performance

The OpenShift Container Platform Ingress Controller, or router, is the ingress point for ingress traffic for applications and services that are configured using routes and ingresses.

When evaluating a single HAProxy router performance in terms of HTTP requests handled per second, the performance varies depending on many factors. In particular:

- HTTP keep-alive/close mode

- Route type

- TLS session resumption client support

- Number of concurrent connections per target route

- Number of target routes

- Back end server page size

- Underlying infrastructure (network/SDN solution, CPU, and so on)

While performance in your specific environment will vary, Red Hat lab tests on a public cloud instance of size 4 vCPU/16GB RAM. A single HAProxy router handling 100 routes terminated by backends serving 1kB static pages is able to handle the following number of transactions per second.

In HTTP keep-alive mode scenarios:

Encryption	LoadBalancerService	HostNetwork
none	21515	29622
edge	16743	22913
passthrough	36786	53295
re-encrypt	21583	25198

In HTTP close (no keep-alive) scenarios:

Encryption	LoadBalancerService	HostNetwork
none	5719	8273
edge	2729	4069
passthrough	4121	5344
re-encrypt	2320	2941

The default Ingress Controller configuration was used with the **spec.tuningOptions.threadCount** field set to **4**. Two different endpoint publishing strategies were tested: Load Balancer Service and Host Network. TLS session resumption was used for encrypted routes. With HTTP keep-alive, a single HAProxy router is capable of saturating a 1 Gbit NIC at page sizes as small as 8 kB.

When running on bare metal with modern processors, you can expect roughly twice the performance of the public cloud instance above. This overhead is introduced by the virtualization layer in place on public clouds and holds mostly true for private cloud-based virtualization as well. The following table is a guide to how many applications to use behind the router:

Number of applications	Application type
5-10	static file/web server or caching proxy
100-1000	applications generating dynamic content

In general, HAProxy can support routes for up to 1000 applications, depending on the technology in use. Ingress Controller performance might be limited by the capabilities and performance of the applications behind it, such as language or static versus dynamic content.

Ingress, or router, sharding should be used to serve more routes towards applications and help horizontally scale the routing tier.

8.8.2. Configuring Ingress Controller liveness, readiness, and startup probes

Cluster administrators can configure the timeout values for the kubelet's liveness, readiness, and startup probes for router deployments that are managed by the OpenShift Container Platform Ingress Controller (router). The liveness and readiness probes of the router use the default timeout value of 1 second, which is too brief when networking or runtime performance is severely degraded. Probe timeouts can cause unwanted router restarts that interrupt application connections. The ability to set larger timeout values can reduce the risk of unnecessary and unwanted restarts.

You can update the **timeoutSeconds** value on the **livenessProbe**, **readinessProbe**, and **startupProbe** parameters of the router container.

Parameter	Description
livenessProbe	The **livenessProbe** reports to the kubelet whether a pod is dead and needs to be restarted.
readinessProbe	The **readinessProbe** reports whether a pod is healthy or unhealthy. When the readiness probe reports an unhealthy pod, then the kubelet marks the pod as not ready to accept traffic. Subsequently, the endpoints for that pod are marked as not ready, and this status propagates to the kube-proxy. On cloud platforms with a configured load balancer, the kube-proxy communicates to the cloud load-balancer not to send traffic to the node with that pod.
startupProbe	The **startupProbe** gives the router pod up to 2 minutes to initialize before the kubelet begins sending the router liveness and readiness probes. This initialization time can prevent routers with many routes or endpoints from prematurely restarting.

IMPORTANT

The timeout configuration option is an advanced tuning technique that can be used to work around issues. However, these issues should eventually be diagnosed and possibly a support case or Jira issue opened for any issues that causes probes to time out.

The following example demonstrates how you can directly patch the default router deployment to set a 5-second timeout for the liveness and readiness probes:

```
$ oc -n openshift-ingress patch deploy/router-default --type=strategic --patch='{"spec":{"template":
{"spec":{"containers":[{"name":"router","livenessProbe":{"timeoutSeconds":5},"readinessProbe":
{"timeoutSeconds":5}}]}}}}'
```

Verification

```
$ oc -n openshift-ingress describe deploy/router-default | grep -e Liveness: -e Readiness:
    Liveness:   http-get http://:1936/healthz delay=0s timeout=5s period=10s #success=1 #failure=3
    Readiness:  http-get http://:1936/healthz/ready delay=0s timeout=5s period=10s #success=1
#failure=3
```

8.8.3. Configuring HAProxy reload interval

When you update a route or an endpoint associated with a route, OpenShift Container Platform router updates the configuration for HAProxy. Then, HAProxy reloads the updated configuration for those changes to take effect. When HAProxy reloads, it generates a new process that handles new connections using the updated configuration.

HAProxy keeps the old process running to handle existing connections until those connections are all closed. When old processes have long-lived connections, these processes can accumulate and consume resources.

The default minimum HAProxy reload interval is five seconds. You can configure an Ingress Controller using its **spec.tuningOptions.reloadInterval** field to set a longer minimum reload interval.

> **WARNING**
>
> Setting a large value for the minimum HAProxy reload interval can cause latency in observing updates to routes and their endpoints. To lessen the risk, avoid setting a value larger than the tolerable latency for updates.

Procedure

- Change the minimum HAProxy reload interval of the default Ingress Controller to 15 seconds by running the following command:

  ```
  $ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --
  patch='{"spec":{"tuningOptions":{"reloadInterval":"15s"}}}'
  ```

8.9. POSTINSTALLATION RHOSP NETWORK CONFIGURATION

You can configure some aspects of an OpenShift Container Platform on Red Hat OpenStack Platform (RHOSP) cluster after installation.

8.9.1. Configuring application access with floating IP addresses

After you install OpenShift Container Platform, configure Red Hat OpenStack Platform (RHOSP) to allow application network traffic.

> **NOTE**
>
> You do not need to perform this procedure if you provided values for
> **platform.openstack.apiFloatingIP** and **platform.openstack.ingressFloatingIP** in the
> **install-config.yaml** file, or **os_api_fip** and **os_ingress_fip** in the **inventory.yaml**
> playbook, during installation. The floating IP addresses are already set.

Prerequisites

- OpenShift Container Platform cluster must be installed

- Floating IP addresses are enabled as described in the OpenShift Container Platform on RHOSP installation documentation.

Procedure

After you install the OpenShift Container Platform cluster, attach a floating IP address to the ingress port:

1. Show the port:

   ```
   $ openstack port show <cluster_name>-<cluster_ID>-ingress-port
   ```

2. Attach the port to the IP address:

   ```
   $ openstack floating ip set --port <ingress_port_ID> <apps_FIP>
   ```

3. Add a wildcard **A** record for ***apps.** to your DNS file:

   ```
   *.apps.<cluster_name>.<base_domain>  IN  A  <apps_FIP>
   ```

> **NOTE**
>
> If you do not control the DNS server but want to enable application access for non-production purposes, you can add these hostnames to **/etc/hosts**:
>
> ```
> <apps_FIP> console-openshift-console.apps.<cluster name>.<base domain>
> <apps_FIP> integrated-oauth-server-openshift-authentication.apps.<cluster name>.<base domain>
> <apps_FIP> oauth-openshift.apps.<cluster name>.<base domain>
> <apps_FIP> prometheus-k8s-openshift-monitoring.apps.<cluster name>.<base domain>
> <apps_FIP> <app name>.apps.<cluster name>.<base domain>
> ```

8.9.2. Kuryr ports pools

A Kuryr ports pool maintains a number of ports on standby for pod creation.

Keeping ports on standby minimizes pod creation time. Without ports pools, Kuryr must explicitly request port creation or deletion whenever a pod is created or deleted.

The Neutron ports that Kuryr uses are created in subnets that are tied to namespaces. These pod ports are also added as subports to the primary port of OpenShift Container Platform cluster nodes.

Because Kuryr keeps each namespace in a separate subnet, a separate ports pool is maintained for each namespace-worker pair.

Prior to installing a cluster, you can set the following parameters in the **cluster-network-03-config.yml** manifest file to configure ports pool behavior:

- The **enablePortPoolsPrepopulation** parameter controls pool prepopulation, which forces Kuryr to add Neutron ports to the pools when the first pod that is configured to use the dedicated network for pods is created in a namespace. The default value is **false**.

- The **poolMinPorts** parameter is the minimum number of free ports that are kept in the pool. The default value is **1**.

- The **poolMaxPorts** parameter is the maximum number of free ports that are kept in the pool. A value of **0** disables that upper bound. This is the default setting.
  If your OpenStack port quota is low, or you have a limited number of IP addresses on the pod network, consider setting this option to ensure that unneeded ports are deleted.

- The **poolBatchPorts** parameter defines the maximum number of Neutron ports that can be created at once. The default value is **3**.

8.9.3. Adjusting Kuryr ports pool settings in active deployments on RHOSP

You can use a custom resource (CR) to configure how Kuryr manages Red Hat OpenStack Platform (RHOSP) Neutron ports to control the speed and efficiency of pod creation on a deployed cluster.

Procedure

1. From a command line, open the Cluster Network Operator (CNO) CR for editing:

   ```
   $ oc edit networks.operator.openshift.io cluster
   ```

2. Edit the settings to meet your requirements. The following file is provided as an example:

   ```
   apiVersion: operator.openshift.io/v1
   kind: Network
   metadata:
     name: cluster
   spec:
     clusterNetwork:
     - cidr: 10.128.0.0/14
       hostPrefix: 23
     serviceNetwork:
     - 172.30.0.0/16
     defaultNetwork:
       type: Kuryr
       kuryrConfig:
         enablePortPoolsPrepopulation: false 1
         poolMinPorts: 1 2
         poolBatchPorts: 3 3
         poolMaxPorts: 5 4
   ```

   1 Set **enablePortPoolsPrepopulation** to **true** to make Kuryr create Neutron ports when the first pod that is configured to use the dedicated network for pods is created in a namespace. This setting raises the Neutron ports quota but can reduce the time that is

required to spawn pods. The default value is **false**.

2 Kuryr creates new ports for a pool if the number of free ports in that pool is lower than the value of **poolMinPorts**. The default value is **1**.

3 **poolBatchPorts** controls the number of new ports that are created if the number of free ports is lower than the value of **poolMinPorts**. The default value is **3**.

4 If the number of free ports in a pool is higher than the value of **poolMaxPorts**, Kuryr deletes them until the number matches that value. Setting the value to **0** disables this upper bound, preventing pools from shrinking. The default value is **0**.

3. Save your changes and quit the text editor to commit your changes.

> **IMPORTANT**
>
> Modifying these options on a running cluster forces the kuryr-controller and kuryr-cni pods to restart. As a result, the creation of new pods and services will be delayed.

8.9.4. Enabling OVS hardware offloading

For clusters that run on Red Hat OpenStack Platform (RHOSP), you can enable Open vSwitch (OVS) hardware offloading.

OVS is a multi-layer virtual switch that enables large-scale, multi-server network virtualization.

Prerequisites

- You installed a cluster on RHOSP that is configured for single-root input/output virtualization (SR-IOV).

- You installed the SR-IOV Network Operator on your cluster.

- You created two **hw-offload** type virtual function (VF) interfaces on your cluster.

> **NOTE**
>
> Application layer gateway flows are broken in OpenShift Container Platform version 4.10, 4.11, and 4.12. Also, you cannot offload the application layer gateway flow for OpenShift Container Platform version 4.13.

Procedure

1. Create an **SriovNetworkNodePolicy** policy for the two **hw-offload** type VF interfaces that are on your cluster:

   The first virtual function interface

   ```
   apiVersion: sriovnetwork.openshift.io/v1
   kind: SriovNetworkNodePolicy 1
   metadata:
     name: "hwoffload9"
     namespace: openshift-sriov-network-operator
   spec:
   ```

```
  deviceType: netdevice
  isRdma: true
  nicSelector:
    pfNames: 2
    - ens6
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: 'true'
  numVfs: 1
  priority: 99
  resourceName: "hwoffload9"
```

1   Insert the **SriovNetworkNodePolicy** value here.

2   Both interfaces must include physical function (PF) names.

The second virtual function interface

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy 1
metadata:
  name: "hwoffload10"
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  isRdma: true
  nicSelector:
    pfNames: 2
    - ens5
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: 'true'
  numVfs: 1
  priority: 99
  resourceName: "hwoffload10"
```

1   Insert the **SriovNetworkNodePolicy** value here.

2   Both interfaces must include physical function (PF) names.

2. Create **NetworkAttachmentDefinition** resources for the two interfaces:

A **NetworkAttachmentDefinition** resource for the first interface

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    k8s.v1.cni.cncf.io/resourceName: openshift.io/hwoffload9
  name: hwoffload9
  namespace: default
spec:
  config: '{ "cniVersion":"0.3.1", "name":"hwoffload9","type":"host-device","device":"ens6"
  }'
```

A **NetworkAttachmentDefinition** resource for the second interface

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    k8s.v1.cni.cncf.io/resourceName: openshift.io/hwoffload10
  name: hwoffload10
  namespace: default
spec:
    config: '{ "cniVersion":"0.3.1", "name":"hwoffload10","type":"host-device","device":"ens5"
    }'
```

3. Use the interfaces that you created with a pod. For example:

   A pod that uses the two OVS offload interfaces

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-testpmd
  namespace: default
  annotations:
    irq-load-balancing.crio.io: disable
    cpu-quota.crio.io: disable
    k8s.v1.cni.cncf.io/resourceName: openshift.io/hwoffload9
    k8s.v1.cni.cncf.io/resourceName: openshift.io/hwoffload10
spec:
  restartPolicy: Never
  containers:
  - name: dpdk-testpmd
    image: quay.io/krister/centos8_nfv-container-dpdk-testpmd:latest
```

8.9.5. Attaching an OVS hardware offloading network

You can attach an Open vSwitch (OVS) hardware offloading network to your cluster.

Prerequisites

- Your cluster is installed and running.

- You provisioned an OVS hardware offloading network on Red Hat OpenStack Platform (RHOSP) to use with your cluster.

Procedure

1. Create a file named **network.yaml** from the following template:

```
spec:
  additionalNetworks:
  - name: hwoffload1
    namespace: cnf
```

```
rawCNIConfig: '{ "cniVersion": "0.3.1", "name": "hwoffload1", "type": "host-
device","pciBusId": "0000:00:05.0", "ipam": {}}' 1
type: Raw
```

where:

pciBusId

Specifies the device that is connected to the offloading network. If you do not have it, you can find this value by running the following command:

```
$ oc describe SriovNetworkNodeState -n openshift-sriov-network-operator
```

2. From a command line, enter the following command to patch your cluster with the file:

```
$ oc apply -f network.yaml
```

8.9.6. Enabling IPv6 connectivity to pods on RHOSP

To enable IPv6 connectivity between pods that have additional networks that are on different nodes, disable port security for the IPv6 port of the server. Disabling port security obviates the need to create allowed address pairs for each IPv6 address that is assigned to pods and enables traffic on the security group.

> **IMPORTANT**
>
> Only the following IPv6 additional network configurations are supported:
>
> - SLAAC and host-device
>
> - SLAAC and MACVLAN
>
> - DHCP stateless and host-device
>
> - DHCP stateless and MACVLAN

Procedure

- On a command line, enter the following command:

```
$ openstack port set --no-security-group --disable-port-security <compute_ipv6_port>
```

> **IMPORTANT**
>
> This command removes security groups from the port and disables port security. Traffic restrictions are removed entirely from the port.

where:

<compute_ipv6_port>

Specifies the IPv6 port of the compute server.

8.9.7. Adding IPv6 connectivity to pods on RHOSP

After you enable IPv6 connectivity in pods, add connectivity to them by using a Container Network Interface (CNI) configuration.

Procedure

1. To edit the Cluster Network Operator (CNO), enter the following command:

```
$ oc edit networks.operator.openshift.io cluster
```

2. Specify your CNI configuration under the **spec** field. For example, the following configuration uses a SLAAC address mode with MACVLAN:

```
...
spec:
  additionalNetworks:
  - name: ipv6
    namespace: ipv6 ❶
    rawCNIConfig: '{ "cniVersion": "0.3.1", "name": "ipv6", "type": "macvlan", "master": "ens4"}' ❷
    type: Raw
```

❶ ❶ Be sure to create pods in the same namespace.

❷ The interface in the network attachment **"master"** field can differ from **"ens4"** when more networks are configured or when a different kernel driver is used.

> **NOTE**
>
> If you are using stateful address mode, include the IP Address Management (IPAM) in the CNI configuration.
>
> DHCPv6 is not supported by Multus.

3. Save your changes and quit the text editor to commit your changes.

Verification

- On a command line, enter the following command:

```
$ oc get network-attachment-definitions -A
```

Example output

```
NAMESPACE     NAME       AGE
ipv6          ipv6       21h
```

You can now create pods that have secondary IPv6 connections.

Additional resources

- Configuration for an additional network attachment

8.9.8. Create pods that have IPv6 connectivity on RHOSP

After you enable IPv6 connectivty for pods and add it to them, create pods that have secondary IPv6 connections.

Procedure

1. Define pods that use your IPv6 namespace and the annotation **k8s.v1.cni.cncf.io/networks: <additional_network_name>**, where **<additional_network_name>** is the name of the additional network. For example, as part of a **Deployment** object:

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: hello-openshift
     namespace: ipv6
   spec:
     affinity:
       podAntiAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
           - labelSelector:
             matchExpressions:
             - key: app
               operator: In
               values:
               - hello-openshift
     replicas: 2
     selector:
       matchLabels:
         app: hello-openshift
     template:
       metadata:
         labels:
           app: hello-openshift
         annotations:
           k8s.v1.cni.cncf.io/networks: ipv6
       spec:
         securityContext:
           runAsNonRoot: true
           seccompProfile:
             type: RuntimeDefault
         containers:
         - name: hello-openshift
           securityContext:
             allowPrivilegeEscalation: false
             capabilities:
               drop:
               - ALL
           image: quay.io/openshift/origin-hello-openshift
           ports:
           - containerPort: 8080
   ```

2. Create the pod. For example, on a command line, enter the following command:

```
$ oc create -f <ipv6_enabled_resource>
```

where:

<ipv6_enabled_resource>

Specifies the file that contains your resource definition.

CHAPTER 9. POSTINSTALLATION STORAGE CONFIGURATION

After installing OpenShift Container Platform, you can further expand and customize your cluster to your requirements, including storage configuration.

9.1. DYNAMIC PROVISIONING

9.1.1. About dynamic provisioning

The **StorageClass** resource object describes and classifies storage that can be requested, as well as provides a means for passing parameters for dynamically provisioned storage on demand. **StorageClass** objects can also serve as a management mechanism for controlling different levels of storage and access to the storage. Cluster Administrators (**cluster-admin**) or Storage Administrators (**storage-admin**) define and create the **StorageClass** objects that users can request without needing any detailed knowledge about the underlying storage volume sources.

The OpenShift Container Platform persistent volume framework enables this functionality and allows administrators to provision a cluster with persistent storage. The framework also gives users a way to request those resources without having any knowledge of the underlying infrastructure.

Many storage types are available for use as persistent volumes in OpenShift Container Platform. While all of them can be statically provisioned by an administrator, some types of storage are created dynamically using the built-in provider and plugin APIs.

9.1.2. Available dynamic provisioning plugins

OpenShift Container Platform provides the following provisioner plugins, which have generic implementations for dynamic provisioning that use the cluster's configured provider's API to create new storage resources:

Storage type	Provisioner plugin name	Notes
Red Hat OpenStack Platform (RHOSP) Cinder	**kubernetes.io/cinder**	
RHOSP Manila Container Storage Interface (CSI)	**manila.csi.openstack.org**	Once installed, the OpenStack Manila CSI Driver Operator and ManilaDriver automatically create the required storage classes for all available Manila share types needed for dynamic provisioning.
AWS Elastic Block Store (EBS)	**kubernetes.io/aws-ebs**	For dynamic provisioning when using multiple clusters in different zones, tag each node with **Key=kubernetes.io/cluster/<cluster_name>,Value=<cluster_id>** where **<cluster_name>** and **<cluster_id>** are unique per cluster.

Storage type	Provisioner plugin name	Notes
Azure Disk	**kubernetes.io/azure-disk**	
Azure File	**kubernetes.io/azure-file**	The **persistent-volume-binder** service account requires permissions to create and get secrets to store the Azure storage account and keys.
GCE Persistent Disk (gcePD)	**kubernetes.io/gce-pd**	In multi-zone configurations, it is advisable to run one OpenShift Container Platform cluster per GCE project to avoid PVs from being created in zones where no node in the current cluster exists.
VMware vSphere	**kubernetes.io/vsphere-volume**	

IMPORTANT

Any chosen provisioner plugin also requires configuration for the relevant cloud, host, or third-party provider as per the relevant documentation.

9.2. DEFINING A STORAGE CLASS

StorageClass objects are currently a globally scoped object and must be created by **cluster-admin** or **storage-admin** users.
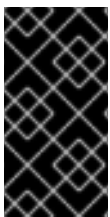
IMPORTANT

The Cluster Storage Operator might install a default storage class depending on the platform in use. This storage class is owned and controlled by the Operator. It cannot be deleted or modified beyond defining annotations and labels. If different behavior is desired, you must define a custom storage class.

The following sections describe the basic definition for a **StorageClass** object and specific examples for each of the supported plugin types.

9.2.1. Basic StorageClass object definition

The following resource shows the parameters and default values that you use to configure a storage class. This example uses the AWS ElasticBlockStore (EBS) object definition.

Sample StorageClass definition

```
kind: StorageClass 1
apiVersion: storage.k8s.io/v1 2
metadata:
  name: <storage-class-name> 3
  annotations: 4
    storageclass.kubernetes.io/is-default-class: 'true'
    ...
provisioner: kubernetes.io/aws-ebs 5
parameters: 6
  type: gp3
...
```

1   (required) The API object type.

2   (required) The current apiVersion.

3   (required) The name of the storage class.

4   (optional) Annotations for the storage class.

5   (required) The type of provisioner associated with this storage class.

6   (optional) The parameters required for the specific provisioner, this will change from plug-in to plug-in.

9.2.2. Storage class annotations

To set a storage class as the cluster-wide default, add the following annotation to your storage class metadata:

```
storageclass.kubernetes.io/is-default-class: "true"
```

For example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
...
```

This enables any persistent volume claim (PVC) that does not specify a specific storage class to automatically be provisioned through the default storage class. However, your cluster can have more than one storage class, but only one of them can be the default storage class.

NOTE

The beta annotation **storageclass.beta.kubernetes.io/is-default-class** is still working; however, it will be removed in a future release.

To set a storage class description, add the following annotation to your storage class metadata:

```
kubernetes.io/description: My Storage Class Description
```

For example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    kubernetes.io/description: My Storage Class Description
...
```

9.2.3. RHOSP Cinder object definition

cinder-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <storage-class-name> 1
provisioner: kubernetes.io/cinder
parameters:
  type: fast 2
  availability: nova 3
  fsType: ext4 4
```

[1] Name of the storage class. The persistent volume claim uses this storage class for provisioning the associated persistent volumes.

[2] Volume type created in Cinder. Default is empty.

[3] Availability Zone. If not specified, volumes are generally round-robined across all active zones where the OpenShift Container Platform cluster has a node.

[4] File system that is created on dynamically provisioned volumes. This value is copied to the **fsType** field of dynamically provisioned persistent volumes and the file system is created when the volume is mounted for the first time. The default value is **ext4**.

9.2.4. AWS Elastic Block Store (EBS) object definition

aws-ebs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
```

```
    name: <storage-class-name> 1
provisioner: kubernetes.io/aws-ebs
parameters:
    type: io1 2
    iopsPerGB: "10" 3
    encrypted: "true" 4
    kmsKeyId: keyvalue 5
    fsType: ext4 6
```

1   (required) Name of the storage class. The persistent volume claim uses this storage class for provisioning the associated persistent volumes.

2   (required) Select from **io1**, **gp3**, **sc1**, **st1**. The default is **gp3**. See the AWS documentation for valid Amazon Resource Name (ARN) values.

3   Optional: Only for **io1** volumes. I/O operations per second per GiB. The AWS volume plugin multiplies this with the size of the requested volume to compute IOPS of the volume. The value cap is 20,000 IOPS, which is the maximum supported by AWS. See the AWS documentation for further details.

4   Optional: Denotes whether to encrypt the EBS volume. Valid values are **true** or **false**.

5   Optional: The full ARN of the key to use when encrypting the volume. If none is supplied, but **encrypted** is set to **true**, then AWS generates a key. See the AWS documentation for a valid ARN value.

6   Optional: File system that is created on dynamically provisioned volumes. This value is copied to the **fsType** field of dynamically provisioned persistent volumes and the file system is created when the volume is mounted for the first time. The default value is **ext4**.

9.2.5. Azure Disk object definition

azure-advanced-disk-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class-name> 1
provisioner: kubernetes.io/azure-disk
volumeBindingMode: WaitForFirstConsumer 2
allowVolumeExpansion: true
parameters:
  kind: Managed 3
  storageaccounttype: Premium_LRS 4
reclaimPolicy: Delete
```

1   Name of the storage class. The persistent volume claim uses this storage class for provisioning the associated persistent volumes.

2   Using **WaitForFirstConsumer** is strongly recommended. This provisions the volume while allowing enough storage to schedule the pod on a free worker node from an available zone.

3   Possible values are **Shared** (default), **Managed**, and **Dedicated**.

> **IMPORTANT**
>
> Red Hat only supports the use of **kind: Managed** in the storage class.
>
> With **Shared** and **Dedicated**, Azure creates unmanaged disks, while OpenShift Container Platform creates a managed disk for machine OS (root) disks. But because Azure Disk does not allow the use of both managed and unmanaged disks on a node, unmanaged disks created with **Shared** or **Dedicated** cannot be attached to OpenShift Container Platform nodes.

4 Azure storage account SKU tier. Default is empty. Note that Premium VMs can attach both **Standard_LRS** and **Premium_LRS** disks, Standard VMs can only attach **Standard_LRS** disks, Managed VMs can only attach managed disks, and unmanaged VMs can only attach unmanaged disks.

a. If **kind** is set to **Shared**, Azure creates all unmanaged disks in a few shared storage accounts in the same resource group as the cluster.

b. If **kind** is set to **Managed**, Azure creates new managed disks.

c. If **kind** is set to **Dedicated** and a **storageAccount** is specified, Azure uses the specified storage account for the new unmanaged disk in the same resource group as the cluster. For this to work:

- The specified storage account must be in the same region.

- Azure Cloud Provider must have write access to the storage account.

d. If **kind** is set to **Dedicated** and a **storageAccount** is not specified, Azure creates a new dedicated storage account for the new unmanaged disk in the same resource group as the cluster.

9.2.6. Azure File object definition

The Azure File storage class uses secrets to store the Azure storage account name and the storage account key that are required to create an Azure Files share. These permissions are created as part of the following procedure.

Procedure

1. Define a **ClusterRole** object that allows access to create and view secrets:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
# name: system:azure-cloud-provider
  name: <persistent-volume-binder-role> 1
rules:
- apiGroups: ['']
  resources: ['secrets']
  verbs:    ['get','create']
```

1 The name of the cluster role to view and create secrets.

2. Add the cluster role to the service account:

```
$ oc adm policy add-cluster-role-to-user <persistent-volume-binder-role>
system:serviceaccount:kube-system:persistent-volume-binder
```

3. Create the Azure File **StorageClass** object:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <azure-file> 1
provisioner: kubernetes.io/azure-file
parameters:
  location: eastus 2
  skuName: Standard_LRS 3
  storageAccount: <storage-account> 4
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

1 Name of the storage class. The persistent volume claim uses this storage class for provisioning the associated persistent volumes.

2 Location of the Azure storage account, such as **eastus**. Default is empty, meaning that a new Azure storage account will be created in the OpenShift Container Platform cluster's location.

3 SKU tier of the Azure storage account, such as **Standard_LRS**. Default is empty, meaning that a new Azure storage account will be created with the **Standard_LRS** SKU.

4 Name of the Azure storage account. If a storage account is provided, then **skuName** and **location** are ignored. If no storage account is provided, then the storage class searches for any storage account that is associated with the resource group for any accounts that match the defined **skuName** and **location**.

9.2.6.1. Considerations when using Azure File

The following file system features are not supported by the default Azure File storage class:

- Symlinks

- Hard links

- Extended attributes

- Sparse files

- Named pipes

Additionally, the owner user identifier (UID) of the Azure File mounted directory is different from the process UID of the container. The **uid** mount option can be specified in the **StorageClass** object to define a specific user identifier to use for the mounted directory.

The following **StorageClass** object demonstrates modifying the user and group identifier, along with enabling symlinks for the mounted directory.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: azure-file
mountOptions:
  - uid=1500 1
  - gid=1500 2
  - mfsymlinks 3
provisioner: kubernetes.io/azure-file
parameters:
  location: eastus
  skuName: Standard_LRS
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

1 Specifies the user identifier to use for the mounted directory.

2 Specifies the group identifier to use for the mounted directory.

3 Enables symlinks.

9.2.7. GCE PersistentDisk (gcePD) object definition

gce-pd-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class-name> 1
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard 2
  replication-type: none
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
reclaimPolicy: Delete
```

1 Name of the storage class. The persistent volume claim uses this storage class for provisioning the associated persistent volumes.

2 Select either **pd-standard** or **pd-ssd**. The default is **pd-standard**.

9.2.8. VMware vSphere object definition

vsphere-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: <storage-class-name> 1
```

```
provisioner: kubernetes.io/vsphere-volume 2
parameters:
  diskformat: thin 3
```

1 Name of the storage class. The persistent volume claim uses this storage class for provisioning the associated persistent volumes.

2 For more information about using VMware vSphere with OpenShift Container Platform, see the VMware vSphere documentation.

3 **diskformat**: **thin**, **zeroedthick** and **eagerzeroedthick** are all valid disk formats. See vSphere docs for additional details regarding the disk format types. The default value is **thin**.

9.2.9. Red Hat Virtualization (RHV) object definition

OpenShift Container Platform creates a default object of type **StorageClass** named **ovirt-csi-sc** which is used for creating dynamically provisioned persistent volumes.

To create additional storage classes for different configurations, create and save a file with the **StorageClass** object described by the following sample YAML:

ovirt-storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage_class_name> 1
  annotations:
    storageclass.kubernetes.io/is-default-class: "<boolean>" 2
provisioner: csi.ovirt.org
allowVolumeExpansion: <boolean> 3
reclaimPolicy: Delete 4
volumeBindingMode: Immediate 5
parameters:
  storageDomainName: <rhv-storage-domain-name> 6
  thinProvisioning: "<boolean>" 7
  csi.storage.k8s.io/fstype: <file_system_type> 8
```

1 Name of the storage class.

2 Set to **false** if the storage class is the default storage class in the cluster. If set to **true**, the existing default storage class must be edited and set to **false**.

3 **true** enables dynamic volume expansion, **false** prevents it. **true** is recommended.

4 Dynamically provisioned persistent volumes of this storage class are created with this reclaim policy. This default policy is **Delete**.

5 Indicates how to provision and bind **PersistentVolumeClaims**. When not set, **VolumeBindingImmediate** is used. This field is only applied by servers that enable the **VolumeScheduling** feature.

6 The RHV storage domain name to use.

7   If **true**, the disk is thin provisioned. If **false**, the disk is preallocated. Thin provisioning is recommended.

8   Optional: File system type to be created. Possible values: **ext4** (default) or **xfs**.

9.3. CHANGING THE DEFAULT STORAGE CLASS

Use this procedure to change the default storage class. For example, you have two defined storage classes, **gp3** and **standard**, and you want to change the default storage class from **gp3** to **standard**.

Procedure

1. List the storage classes:

   ```
   $ oc get storageclass
   ```

   Example output

   ```
   NAME             TYPE
   gp3 (default)      kubernetes.io/aws-ebs   ①
   standard           kubernetes.io/aws-ebs
   ```

   1   **(default)** denotes the default storage class.

2. Change the value of the **storageclass.kubernetes.io/is-default-class** annotation to **false** for the default storage class:

   ```
   $ oc patch storageclass gp3 -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "false"}}}'
   ```

3. Make another storage class the default by setting the **storageclass.kubernetes.io/is-default-class** annotation to **true**:

   ```
   $ oc patch storageclass standard -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "true"}}}'
   ```

4. Verify the changes:

   ```
   $ oc get storageclass
   ```

   Example output

   ```
   NAME               TYPE
   gp3                kubernetes.io/aws-ebs
   standard (default)   kubernetes.io/aws-ebs
   ```

9.4. OPTIMIZING STORAGE

Optimizing storage helps to minimize storage use across all resources. By optimizing storage, administrators help ensure that existing storage resources are working in an efficient manner.

9.5. AVAILABLE PERSISTENT STORAGE OPTIONS

Understand your persistent storage options so that you can optimize your OpenShift Container Platform environment.

Table 9.1. Available storage options

Storage type	Description	Examples
Block	- Presented to the operating system (OS) as a block device - Suitable for applications that need full control of storage and operate at a low level on files bypassing the file system - Also referred to as a Storage Area Network (SAN) - Non-shareable, which means that only one client at a time can mount an endpoint of this type	AWS EBS and VMware vSphere support dynamic persistent volume (PV) provisioning natively in OpenShift Container Platform.
File	- Presented to the OS as a file system export to be mounted - Also referred to as Network Attached Storage (NAS) - Concurrency, latency, file locking mechanisms, and other capabilities vary widely between protocols, implementations, vendors, and scales.	RHEL NFS, NetApp NFS [1], and Vendor NFS
Object	- Accessible through a REST API endpoint - Configurable for use in the OpenShift image registry - Applications must build their drivers into the application and/or container.	AWS S3

1. NetApp NFS supports dynamic PV provisioning when using the Trident plugin.

9.6. RECOMMENDED CONFIGURABLE STORAGE TECHNOLOGY

The following table summarizes the recommended and configurable storage technologies for the given OpenShift Container Platform cluster application.

Table 9.2. Recommended and configurable storage technology

Storage type	Block	File	Object
ROX[1]	Yes[4]	Yes[4]	Yes
RWX[2]	No	Yes	Yes
Registry	Configurable	Configurable	Recommended
Scaled registry	Not configurable	Configurable	Recommended
Metrics[3]	Recommended	Configurable[5]	Not configurable
Elasticsearch Logging	Recommended	Configurable[6]	Not supported[6]
Loki Logging	Not configurable	Not configurable	Recommended
Apps	Recommended	Recommended	Not configurable[7]

[1] **ReadOnlyMany**

[2] **ReadWriteMany**

[3] Prometheus is the underlying technology used for metrics.

[4] This does not apply to physical disk, VM physical disk, VMDK, loopback over NFS, AWS EBS, and Azure Disk.

[5] For metrics, using file storage with the **ReadWriteMany** (RWX) access mode is unreliable. If you use file storage, do not configure the RWX access mode on any persistent volume claims (PVCs) that are configured for use with metrics.

[6] For logging, review the recommended storage solution in Configuring persistent storage for the log store section. Using NFS storage as a persistent volume or through NAS, such as Gluster, can corrupt the data. Hence, NFS is not supported for Elasticsearch storage and LokiStack log store in OpenShift Container Platform Logging. You must use one persistent volume type per log store.

[7] Object storage is not consumed through OpenShift Container Platform's PVs or PVCs. Apps must integrate with the object storage REST API.

> **NOTE**
>
> A scaled registry is an OpenShift image registry where two or more pod replicas are running.

9.6.1. Specific application storage recommendations

> **IMPORTANT**
>
> Testing shows issues with using the NFS server on Red Hat Enterprise Linux (RHEL) as storage backend for core services. This includes the OpenShift Container Registry and Quay, Prometheus for monitoring storage, and Elasticsearch for logging storage. Therefore, using RHEL NFS to back PVs used by core services is not recommended.
>
> Other NFS implementations on the marketplace might not have these issues. Contact the individual NFS implementation vendor for more information on any testing that was possibly completed against these OpenShift Container Platform core components.

9.6.1.1. Registry

In a non-scaled/high-availability (HA) OpenShift image registry cluster deployment:

- The storage technology does not have to support RWX access mode.

- The storage technology must ensure read-after-write consistency.

- The preferred storage technology is object storage followed by block storage.

- File storage is not recommended for OpenShift image registry cluster deployment with production workloads.

9.6.1.2. Scaled registry

In a scaled/HA OpenShift image registry cluster deployment:

- The storage technology must support RWX access mode.

- The storage technology must ensure read-after-write consistency.

- The preferred storage technology is object storage.

- Red Hat OpenShift Data Foundation (ODF), Amazon Simple Storage Service (Amazon S3), Google Cloud Storage (GCS), Microsoft Azure Blob Storage, and OpenStack Swift are supported.

- Object storage should be S3 or Swift compliant.

- For non-cloud platforms, such as vSphere and bare metal installations, the only configurable technology is file storage.

- Block storage is not configurable.

9.6.1.3. Metrics

In an OpenShift Container Platform hosted metrics cluster deployment:

- The preferred storage technology is block storage.

- Object storage is not configurable.
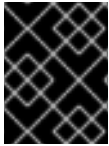
> **IMPORTANT**
>
> It is not recommended to use file storage for a hosted metrics cluster deployment with production workloads.

9.6.1.4. Logging

In an OpenShift Container Platform hosted logging cluster deployment:

- Loki Operator:

  - The preferred storage technology is S3 compatible Object storage.

  - Block storage is not configurable.

- OpenShift Elasticsearch Operator:

  - The preferred storage technology is block storage.

  - Object storage is not supported.

> **NOTE**
>
> As of logging version 5.4.3 the OpenShift Elasticsearch Operator is deprecated and is planned to be removed in a future release. Red Hat will provide bug fixes and support for this feature during the current release lifecycle, but this feature will no longer receive enhancements and will be removed. As an alternative to using the OpenShift Elasticsearch Operator to manage the default log storage, you can use the Loki Operator.

9.6.1.5. Applications

Application use cases vary from application to application, as described in the following examples:

- Storage technologies that support dynamic PV provisioning have low mount time latencies, and are not tied to nodes to support a healthy cluster.

- Application developers are responsible for knowing and understanding the storage requirements for their application, and how it works with the provided storage to ensure that issues do not occur when an application scales or interacts with the storage layer.

9.6.2. Other specific application storage recommendations

> **IMPORTANT**
>
> It is not recommended to use RAID configurations on **Write** intensive workloads, such as **etcd**. If you are running **etcd** with a RAID configuration, you might be at risk of encountering performance issues with your workloads.

- Red Hat OpenStack Platform (RHOSP) Cinder: RHOSP Cinder tends to be adept in ROX access mode use cases.

- Databases: Databases (RDBMSs, NoSQL DBs, etc.) tend to perform best with dedicated block storage.

- The etcd database must have enough storage and adequate performance capacity to enable a large cluster. Information about monitoring and benchmarking tools to establish ample storage and a high-performance environment is described in *Recommended etcd practices*.

Additional resources

- [Recommended etcd practices](#)

9.7. DEPLOY RED HAT OPENSHIFT DATA FOUNDATION

Red Hat OpenShift Data Foundation is a provider of agnostic persistent storage for OpenShift Container Platform supporting file, block, and object storage, either in-house or in hybrid clouds. As a Red Hat storage solution, Red Hat OpenShift Data Foundation is completely integrated with OpenShift Container Platform for deployment, management, and monitoring.

If you are looking for Red Hat OpenShift Data Foundation information about...	See the following Red Hat OpenShift Data Foundation documentation:
What's new, known issues, notable bug fixes, and Technology Previews	[OpenShift Data Foundation 4.12 Release Notes](#)
Supported workloads, layouts, hardware and software requirements, sizing and scaling recommendations	[Planning your OpenShift Data Foundation 4.12 deployment](#)
Instructions on deploying OpenShift Data Foundation to use an external Red Hat Ceph Storage cluster	[Deploying OpenShift Data Foundation 4.12 in external mode](#)
Instructions on deploying OpenShift Data Foundation to local storage on bare metal infrastructure	[Deploying OpenShift Data Foundation 4.12 using bare metal infrastructure](#)
Instructions on deploying OpenShift Data Foundation on Red Hat OpenShift Container Platform VMware vSphere clusters	[Deploying OpenShift Data Foundation 4.12 on VMware vSphere](#)
Instructions on deploying OpenShift Data Foundation using Amazon Web Services for local or cloud storage	[Deploying OpenShift Data Foundation 4.12 using Amazon Web Services](#)
Instructions on deploying and managing OpenShift Data Foundation on existing Red Hat OpenShift Container Platform Google Cloud clusters	[Deploying and managing OpenShift Data Foundation 4.12 using Google Cloud](#)
Instructions on deploying and managing OpenShift Data Foundation on existing Red Hat OpenShift Container Platform Azure clusters	[Deploying and managing OpenShift Data Foundation 4.12 using Microsoft Azure](#)

If you are looking for Red Hat OpenShift Data Foundation information about...	See the following Red Hat OpenShift Data Foundation documentation:
Instructions on deploying OpenShift Data Foundation to use local storage on IBM Power infrastructure	Deploying OpenShift Data Foundation on IBM Power
Instructions on deploying OpenShift Data Foundation to use local storage on IBM Z infrastructure	Deploying OpenShift Data Foundation on IBM Z infrastructure
Allocating storage to core services and hosted applications in Red Hat OpenShift Data Foundation, including snapshot and clone	Managing and allocating resources
Managing storage resources across a hybrid cloud or multicloud environment using the Multicloud Object Gateway (NooBaa)	Managing hybrid and multicloud resources
Safely replacing storage devices for Red Hat OpenShift Data Foundation	Replacing devices
Safely replacing a node in a Red Hat OpenShift Data Foundation cluster	Replacing nodes
Scaling operations in Red Hat OpenShift Data Foundation	Scaling storage
Monitoring a Red Hat OpenShift Data Foundation 4.12 cluster	Monitoring Red Hat OpenShift Data Foundation 4.12
Resolve issues encountered during operations	Troubleshooting OpenShift Data Foundation 4.12
Migrating your OpenShift Container Platform cluster from version 3 to version 4	Migration

9.8. ADDITIONAL RESOURCES

- Configuring the Elasticsearch log store

CHAPTER 10. PREPARING FOR USERS

After installing OpenShift Container Platform, you can further expand and customize your cluster to your requirements, including taking steps to prepare for users.

10.1. UNDERSTANDING IDENTITY PROVIDER CONFIGURATION

The OpenShift Container Platform control plane includes a built-in OAuth server. Developers and administrators obtain OAuth access tokens to authenticate themselves to the API.

As an administrator, you can configure OAuth to specify an identity provider after you install your cluster.

10.1.1. About identity providers in OpenShift Container Platform

By default, only a **kubeadmin** user exists on your cluster. To specify an identity provider, you must create a custom resource (CR) that describes that identity provider and add it to the cluster.

> **NOTE**
>
> OpenShift Container Platform user names containing /, **:**, and **%** are not supported.

10.1.2. Supported identity providers

You can configure the following types of identity providers:

Identity provider	Description
htpasswd	Configure the **htpasswd** identity provider to validate user names and passwords against a flat file generated using **htpasswd**.
Keystone	Configure the **keystone** identity provider to integrate your OpenShift Container Platform cluster with Keystone to enable shared authentication with an OpenStack Keystone v3 server configured to store users in an internal database.
LDAP	Configure the **ldap** identity provider to validate user names and passwords against an LDAPv3 server, using simple bind authentication.
Basic authentication	Configure a **basic-authentication** identity provider for users to log in to OpenShift Container Platform with credentials validated against a remote identity provider. Basic authentication is a generic backend integration mechanism.
Request header	Configure a **request-header** identity provider to identify users from request header values, such as **X-Remote-User**. It is typically used in combination with an authenticating proxy, which sets the request header value.
GitHub or GitHub Enterprise	Configure a **github** identity provider to validate user names and passwords against GitHub or GitHub Enterprise's OAuth authentication server.

Identity provider	Description
GitLab	Configure a **gitlab** identity provider to use GitLab.com or any other GitLab instance as an identity provider.
Google	Configure a **google** identity provider using Google's OpenID Connect integration.
OpenID Connect	Configure an **oidc** identity provider to integrate with an OpenID Connect identity provider using an Authorization Code Flow.

After you define an identity provider, you can use RBAC to define and apply permissions .

10.1.3. Identity provider parameters

The following parameters are common to all identity providers:

Parameter	Description
name	The provider name is prefixed to provider user names to form an identity name.
mappingMethod	Defines how new identities are mapped to users when they log in. Enter one of the following values: **claim** The default value. Provisions a user with the identity's preferred user name. Fails if a user with that user name is already mapped to another identity. **lookup** Looks up an existing identity, user identity mapping, and user, but does not automatically provision users or identities. This allows cluster administrators to set up identities and users manually, or using an external process. Using this method requires you to manually provision users. **add** Provisions a user with the identity's preferred user name. If a user with that user name already exists, the identity is mapped to the existing user, adding to any existing identity mappings for the user. Required when multiple identity providers are configured that identify the same set of users and map to the same user names.

> **NOTE**
>
> When adding or changing identity providers, you can map identities from the new provider to existing users by setting the **mappingMethod** parameter to **add**.

10.1.4. Sample identity provider CR

The following custom resource (CR) shows the parameters and default values that you use to configure an identity provider. This example uses the htpasswd identity provider.

Sample identity provider CR

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

```
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: my_identity_provider 1
    mappingMethod: claim 2
    type: HTPasswd
    htpasswd:
      fileData:
        name: htpass-secret 3
```

1 This provider name is prefixed to provider user names to form an identity name.

2 Controls how mappings are established between this provider's identities and **User** objects.

3 An existing secret containing a file generated using **htpasswd**.

10.2. USING RBAC TO DEFINE AND APPLY PERMISSIONS

Understand and apply role-based access control.

10.2.1. RBAC overview

Role-based access control (RBAC) objects determine whether a user is allowed to perform a given action within a project.

Cluster administrators can use the cluster roles and bindings to control who has various access levels to the OpenShift Container Platform platform itself and all projects.

Developers can use local roles and bindings to control who has access to their projects. Note that authorization is a separate step from authentication, which is more about determining the identity of who is taking the action.

Authorization is managed using:

Authorization object	Description
Rules	Sets of permitted verbs on a set of objects. For example, whether a user or service account can **create** pods.
Roles	Collections of rules. You can associate, or bind, users and groups to multiple roles.
Bindings	Associations between users and/or groups with a role.

There are two levels of RBAC roles and bindings that control authorization:

RBAC level	Description
Cluster RBAC	Roles and bindings that are applicable across all projects. *Cluster roles* exist cluster-wide, and *cluster role bindings* can reference only cluster roles.
Local RBAC	Roles and bindings that are scoped to a given project. While *local roles* exist only in a single project, local role bindings can reference *both* cluster and local roles.

A cluster role binding is a binding that exists at the cluster level. A role binding exists at the project level. The cluster role *view* must be bound to a user using a local role binding for that user to view the project. Create local roles only if a cluster role does not provide the set of permissions needed for a particular situation.

This two-level hierarchy allows reuse across multiple projects through the cluster roles while allowing customization inside of individual projects through local roles.

During evaluation, both the cluster role bindings and the local role bindings are used. For example:

1. Cluster-wide "allow" rules are checked.

2. Locally-bound "allow" rules are checked.

3. Deny by default.

10.2.1.1. Default cluster roles

OpenShift Container Platform includes a set of default cluster roles that you can bind to users and groups cluster-wide or locally.

IMPORTANT

It is not recommended to manually modify the default cluster roles. Modifications to these system roles can prevent a cluster from functioning properly.

Default cluster role	Description
admin	A project manager. If used in a local binding, an **admin** has rights to view any resource in the project and modify any resource in the project except for quota.
basic-user	A user that can get basic information about projects and users.
cluster-admin	A super-user that can perform any action in any project. When bound to a user with a local binding, they have full control over quota and every action on every resource in the project.
cluster-status	A user that can get basic cluster status information.
cluster-reader	A user that can get or view most of the objects but cannot modify them.

Default cluster role	Description
edit	A user that can modify most objects in a project but does not have the power to view or modify roles or bindings.
self-provisioner	A user that can create their own projects.
view	A user who cannot make any modifications, but can see most objects in a project. They cannot view or modify roles or bindings.

Be mindful of the difference between local and cluster bindings. For example, if you bind the **cluster-admin** role to a user by using a local role binding, it might appear that this user has the privileges of a cluster administrator. This is not the case. Binding the **cluster-admin** to a user in a project grants super administrator privileges for only that project to the user. That user has the permissions of the cluster role **admin**, plus a few additional permissions like the ability to edit rate limits, for that project. This binding can be confusing via the web console UI, which does not list cluster role bindings that are bound to true cluster administrators. However, it does list local role bindings that you can use to locally bind **cluster-admin**.

The relationships between cluster roles, local roles, cluster role bindings, local role bindings, users, groups and service accounts are illustrated below.
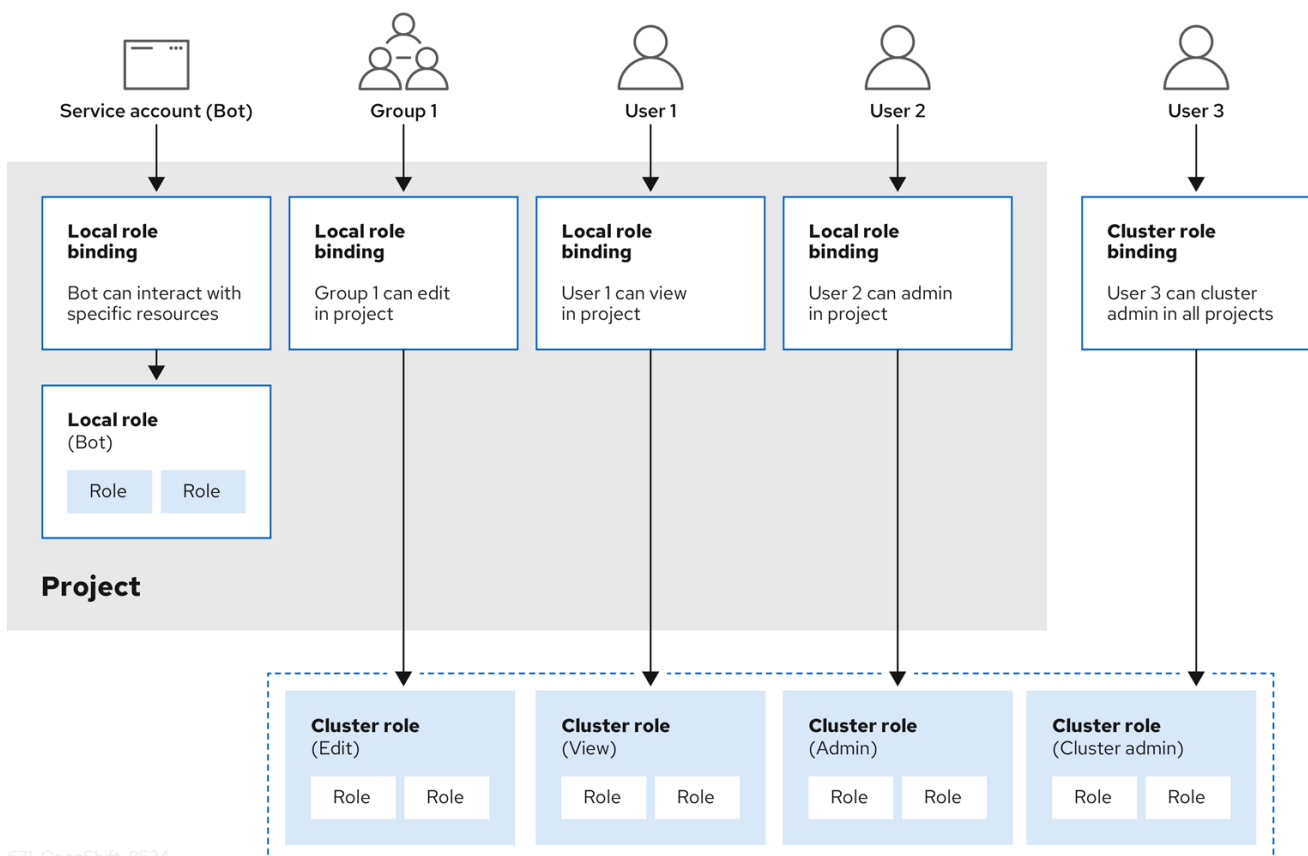
671_OpenShift_0524

> **WARNING**
>
> The **get pods/exec**, **get pods/***, and **get *** rules grant execution privileges when they are applied to a role. Apply the principle of least privilege and assign only the minimal RBAC rights required for users and agents. For more information, see [RBAC rules allow execution privileges](#) .

10.2.1.2. Evaluating authorization

OpenShift Container Platform evaluates authorization by using:

Identity

> The user name and list of groups that the user belongs to.

Action

> The action you perform. In most cases, this consists of:
>
> - **Project**: The project you access. A project is a Kubernetes namespace with additional annotations that allows a community of users to organize and manage their content in isolation from other communities.
>
> - **Verb** : The action itself: **get**, **list**, **create**, **update**, **delete**, **deletecollection**, or **watch**.
>
> - **Resource name**: The API endpoint that you access.

Bindings

> The full list of bindings, the associations between users or groups with a role.

OpenShift Container Platform evaluates authorization by using the following steps:

1. The identity and the project-scoped action is used to find all bindings that apply to the user or their groups.

2. Bindings are used to locate all the roles that apply.

3. Roles are used to find all the rules that apply.

4. The action is checked against each rule to find a match.

5. If no matching rule is found, the action is then denied by default.

TIP

Remember that users and groups can be associated with, or bound to, multiple roles at the same time.

Project administrators can use the CLI to view local roles and bindings, including a matrix of the verbs and resources each are associated with.

> **IMPORTANT**
>
> The cluster role bound to the project administrator is limited in a project through a local binding. It is not bound cluster-wide like the cluster roles granted to the **cluster-admin** or **system:admin**.
>
> Cluster roles are roles defined at the cluster level but can be bound either at the cluster level or at the project level.

10.2.1.2.1. Cluster role aggregation

The default admin, edit, view, and cluster-reader cluster roles support cluster role aggregation, where the cluster rules for each role are dynamically updated as new rules are created. This feature is relevant only if you extend the Kubernetes API by creating custom resources.

10.2.2. Projects and namespaces

A Kubernetes *namespace* provides a mechanism to scope resources in a cluster. The Kubernetes documentation has more information on namespaces.

Namespaces provide a unique scope for:

- Named resources to avoid basic naming collisions.

- Delegated management authority to trusted users.

- The ability to limit community resource consumption.

Most objects in the system are scoped by namespace, but some are excepted and have no namespace, including nodes and users.

A *project* is a Kubernetes namespace with additional annotations and is the central vehicle by which access to resources for regular users is managed. A project allows a community of users to organize and manage their content in isolation from other communities. Users must be given access to projects by administrators, or if allowed to create projects, automatically have access to their own projects.

Projects can have a separate **name**, **displayName**, and **description**.

- The mandatory **name** is a unique identifier for the project and is most visible when using the CLI tools or API. The maximum name length is 63 characters.

- The optional **displayName** is how the project is displayed in the web console (defaults to **name**).

- The optional **description** can be a more detailed description of the project and is also visible in the web console.

Each project scopes its own set of:

Object	Description
Objects	Pods, services, replication controllers, etc.
Policies	Rules for which users can or cannot perform actions on objects.

Object	Description
Constraints	Quotas for each kind of object that can be limited.
Service accounts	Service accounts act automatically with designated access to objects in the project.

Cluster administrators can create projects and delegate administrative rights for the project to any member of the user community. Cluster administrators can also allow developers to create their own projects.

Developers and administrators can interact with projects by using the CLI or the web console.

10.2.3. Default projects

OpenShift Container Platform comes with a number of default projects, and projects starting with **openshift-** are the most essential to users. These projects host master components that run as pods and other infrastructure components. The pods created in these namespaces that have a critical pod annotation are considered critical, and the have guaranteed admission by kubelet. Pods created for master components in these namespaces are already marked as critical.

> **NOTE**
>
> You cannot assign an SCC to pods created in one of the default namespaces: **default**, **kube-system**, **kube-public**, **openshift-node**, **openshift-infra**, and **openshift**. You cannot use these namespaces for running pods or services.

10.2.4. Viewing cluster roles and bindings

You can use the **oc** CLI to view cluster roles and bindings by using the **oc describe** command.

Prerequisites

- Install the **oc** CLI.

- Obtain permission to view the cluster roles and bindings.

Users with the **cluster-admin** default cluster role bound cluster-wide can perform any action on any resource, including viewing cluster roles and bindings.

Procedure

1. To view the cluster roles and their associated rule sets:

   ```
   $ oc describe clusterrole.rbac
   ```

   Example output

   ```
   Name:         admin
   Labels:       kubernetes.io/bootstrapping=rbac-defaults
   Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
   ```

```
PolicyRule:
  Resources                                   Non-Resource URLs  Resource Names  Verbs
  ---------                                   -----------------  --------------  -----
  .packages.apps.redhat.com                        []                []         [* create update
patch delete get list watch]
  imagestreams                                     []                []         [create delete
deletecollection get list patch update watch create get list watch]
  imagestreams.image.openshift.io                  []                []         [create delete
deletecollection get list patch update watch create get list watch]
  secrets                                          []                []         [create delete deletecollection
get list patch update watch get list watch create delete deletecollection patch update]
  buildconfigs/webhooks                            []                []         [create delete
deletecollection get list patch update watch get list watch]
  buildconfigs                                     []                []         [create delete
deletecollection get list patch update watch get list watch]
  buildlogs                                        []                []         [create delete deletecollection
get list patch update watch get list watch]
  deploymentconfigs/scale                          []                []         [create delete
deletecollection get list patch update watch get list watch]
  deploymentconfigs                                []                []         [create delete
deletecollection get list patch update watch get list watch]
  imagestreamimages                                []                []         [create delete
deletecollection get list patch update watch get list watch]
  imagestreammappings                              []                []         [create delete
deletecollection get list patch update watch get list watch]
  imagestreamtags                                  []                []         [create delete
deletecollection get list patch update watch get list watch]
  processedtemplates                               []                []         [create delete
deletecollection get list patch update watch get list watch]
  routes                                           []                []         [create delete deletecollection
get list patch update watch get list watch]
  templateconfigs                                  []                []         [create delete
deletecollection get list patch update watch get list watch]
  templateinstances                                []                []         [create delete
deletecollection get list patch update watch get list watch]
  templates                                        []                []         [create delete
deletecollection get list patch update watch get list watch]
  deploymentconfigs.apps.openshift.io/scale        []                []         [create delete
deletecollection get list patch update watch get list watch]
  deploymentconfigs.apps.openshift.io              []                []         [create delete
deletecollection get list patch update watch get list watch]
  buildconfigs.build.openshift.io/webhooks         []                []         [create delete
deletecollection get list patch update watch get list watch]
  buildconfigs.build.openshift.io                  []                []         [create delete
deletecollection get list patch update watch get list watch]
  buildlogs.build.openshift.io                     []                []         [create delete
deletecollection get list patch update watch get list watch]
  imagestreamimages.image.openshift.io             []                []         [create delete
deletecollection get list patch update watch get list watch]
  imagestreammappings.image.openshift.io           []                []         [create delete
deletecollection get list patch update watch get list watch]
  imagestreamtags.image.openshift.io               []                []         [create delete
deletecollection get list patch update watch get list watch]
  routes.route.openshift.io                        []                []         [create delete
deletecollection get list patch update watch get list watch]
  processedtemplates.template.openshift.io         []                []         [create delete
```

```
deletecollection get list patch update watch get list watch]
  templateconfigs.template.openshift.io              []            []            [create delete
deletecollection get list patch update watch get list watch]
  templateinstances.template.openshift.io            []            []            [create delete
deletecollection get list patch update watch get list watch]
  templates.template.openshift.io                    []            []            [create delete
deletecollection get list patch update watch get list watch]
  serviceaccounts                                    []            []            [create delete
deletecollection get list patch update watch impersonate create delete deletecollection patch
update get list watch]
  imagestreams/secrets                               []            []            [create delete
deletecollection get list patch update watch]
  rolebindings                                       []            []            [create delete
deletecollection get list patch update watch]
  roles                                              []            []            [create delete deletecollection
get list patch update watch]
  rolebindings.authorization.openshift.io            []            []            [create delete
deletecollection get list patch update watch]
  roles.authorization.openshift.io                   []            []            [create delete
deletecollection get list patch update watch]
  imagestreams.image.openshift.io/secrets            []            []            [create delete
deletecollection get list patch update watch]
  rolebindings.rbac.authorization.k8s.io             []            []            [create delete
deletecollection get list patch update watch]
  roles.rbac.authorization.k8s.io                    []            []            [create delete
deletecollection get list patch update watch]
  networkpolicies.extensions                         []            []            [create delete
deletecollection patch update create delete deletecollection get list patch update watch get
list watch]
  networkpolicies.networking.k8s.io                  []            []            [create delete
deletecollection patch update create delete deletecollection get list patch update watch get
list watch]
  configmaps                                         []            []            [create delete
deletecollection patch update get list watch]
  endpoints                                          []            []            [create delete
deletecollection patch update get list watch]
  persistentvolumeclaims                             []            []            [create delete
deletecollection patch update get list watch]
  pods                                               []            []            [create delete deletecollection
patch update get list watch]
  replicationcontrollers/scale                       []            []            [create delete
deletecollection patch update get list watch]
  replicationcontrollers                             []            []            [create delete
deletecollection patch update get list watch]
  services                                           []            []            [create delete deletecollection
patch update get list watch]
  daemonsets.apps                                    []            []            [create delete
deletecollection patch update get list watch]
  deployments.apps/scale                             []            []            [create delete
deletecollection patch update get list watch]
  deployments.apps                                   []            []            [create delete
deletecollection patch update get list watch]
  replicasets.apps/scale                             []            []            [create delete
deletecollection patch update get list watch]
  replicasets.apps                                   []            []            [create delete
deletecollection patch update get list watch]
```

```
  statefulsets.apps/scale                          []            []           [create delete
deletecollection patch update get list watch]
  statefulsets.apps                                []            []           [create delete
deletecollection patch update get list watch]
  horizontalpodautoscalers.autoscaling             []            []           [create delete
deletecollection patch update get list watch]
  cronjobs.batch                                   []            []           [create delete
deletecollection patch update get list watch]
  jobs.batch                                       []            []           [create delete
deletecollection patch update get list watch]
  daemonsets.extensions                            []            []           [create delete
deletecollection patch update get list watch]
  deployments.extensions/scale                     []            []           [create delete
deletecollection patch update get list watch]
  deployments.extensions                           []            []           [create delete
deletecollection patch update get list watch]
  ingresses.extensions                             []            []           [create delete
deletecollection patch update get list watch]
  replicasets.extensions/scale                     []            []           [create delete
deletecollection patch update get list watch]
  replicasets.extensions                           []            []           [create delete
deletecollection patch update get list watch]
  replicationcontrollers.extensions/scale          []            []           [create delete
deletecollection patch update get list watch]
  poddisruptionbudgets.policy                      []            []           [create delete
deletecollection patch update get list watch]
  deployments.apps/rollback                        []            []           [create delete
deletecollection patch update]
  deployments.extensions/rollback                  []            []           [create delete
deletecollection patch update]
  catalogsources.operators.coreos.com             []            []           [create update
patch delete get list watch]
  clusterserviceversions.operators.coreos.com     []            []           [create update
patch delete get list watch]
  installplans.operators.coreos.com               []            []           [create update
patch delete get list watch]
  packagemanifests.operators.coreos.com           []            []           [create update
patch delete get list watch]
  subscriptions.operators.coreos.com              []            []           [create update
patch delete get list watch]
  buildconfigs/instantiate                         []            []           [create]
  buildconfigs/instantiatebinary                   []            []           [create]
  builds/clone                                     []            []           [create]
  deploymentconfigrollbacks                        []            []           [create]
  deploymentconfigs/instantiate                    []            []           [create]
  deploymentconfigs/rollback                       []            []           [create]
  imagestreamimports                               []            []           [create]
  localresourceaccessreviews                       []            []           [create]
  localsubjectaccessreviews                        []            []           [create]
  podsecuritypolicyreviews                         []            []           [create]
  podsecuritypolicyselfsubjectreviews              []            []           [create]
  podsecuritypolicysubjectreviews                  []            []           [create]
  resourceaccessreviews                            []            []           [create]
  routes/custom-host                               []            []           [create]
  subjectaccessreviews                             []            []           [create]
  subjectrulesreviews                              []            []           [create]
```

```
deploymentconfigrollbacks.apps.openshift.io            []            []            [create]
deploymentconfigs.apps.openshift.io/instantiate        []            []            [create]
deploymentconfigs.apps.openshift.io/rollback           []            []            [create]
localsubjectaccessreviews.authorization.k8s.io         []            []            [create]
localresourceaccessreviews.authorization.openshift.io  []            []            [create]
localsubjectaccessreviews.authorization.openshift.io   []            []            [create]
resourceaccessreviews.authorization.openshift.io       []            []            [create]
subjectaccessreviews.authorization.openshift.io        []            []            [create]
subjectrulesreviews.authorization.openshift.io         []            []            [create]
buildconfigs.build.openshift.io/instantiate            []            []            [create]
buildconfigs.build.openshift.io/instantiatebinary      []            []            [create]
builds.build.openshift.io/clone                        []            []            [create]
imagestreamimports.image.openshift.io                  []            []            [create]
routes.route.openshift.io/custom-host                  []            []            [create]
podsecuritypolicyreviews.security.openshift.io         []            []            [create]
podsecuritypolicyselfsubjectreviews.security.openshift.io []         []            [create]
podsecuritypolicysubjectreviews.security.openshift.io  []            []            [create]
jenkins.build.openshift.io                             []            []            [edit view view admin
edit view]
builds                                                 []            []            [get create delete
deletecollection get list patch update watch get list watch]
builds.build.openshift.io                              []            []            [get create delete
deletecollection get list patch update watch get list watch]
projects                                               []            []            [get delete get delete get patch
update]
projects.project.openshift.io                          []            []            [get delete get delete
get patch update]
namespaces                                             []            []            [get get list watch]
pods/attach                                            []            []            [get list watch create delete
deletecollection patch update]
pods/exec                                              []            []            [get list watch create delete
deletecollection patch update]
pods/portforward                                       []            []            [get list watch create
delete deletecollection patch update]
pods/proxy                                             []            []            [get list watch create delete
deletecollection patch update]
services/proxy                                         []            []            [get list watch create delete
deletecollection patch update]
routes/status                                          []            []            [get list watch update]
routes.route.openshift.io/status                       []            []            [get list watch update]
appliedclusterresourcequotas                           []            []            [get list watch]
bindings                                               []            []            [get list watch]
builds/log                                             []            []            [get list watch]
deploymentconfigs/log                                  []            []            [get list watch]
deploymentconfigs/status                               []            []            [get list watch]
events                                                 []            []            [get list watch]
imagestreams/status                                    []            []            [get list watch]
limitranges                                            []            []            [get list watch]
namespaces/status                                      []            []            [get list watch]
pods/log                                               []            []            [get list watch]
pods/status                                            []            []            [get list watch]
replicationcontrollers/status                          []            []            [get list watch]
resourcequotas/status                                  []            []            [get list watch]
resourcequotas                                         []            []            [get list watch]
resourcequotausages                                    []            []            [get list watch]
rolebindingrestrictions                                []            []            [get list watch]
```

```
  deploymentconfigs.apps.openshift.io/log                []              []               [get list watch]
  deploymentconfigs.apps.openshift.io/status             []              []                [get list watch]
  controllerrevisions.apps                    []            []            [get list watch]
  rolebindingrestrictions.authorization.openshift.io      []              []               [get list watch]
  builds.build.openshift.io/log               []            []            [get list watch]
  imagestreams.image.openshift.io/status                  []              []              [get list watch]
  appliedclusterresourcequotas.quota.openshift.io         []              []               [get list watch]
  imagestreams/layers                         []            []            [get update get]
  imagestreams.image.openshift.io/layers                  []              []              [get update get]
  builds/details                       []           []            [update]
  builds.build.openshift.io/details               []             []               [update]


Name:       basic-user
Labels:     <none>
Annotations:  openshift.io/description: A user that can get basic information about projects.
          rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
 Resources                              Non-Resource URLs  Resource Names  Verbs
  ---------                             ----------------   --------------  -----
  selfsubjectrulesreviews                        []            []            [create]
  selfsubjectaccessreviews.authorization.k8s.io       []             []               [create]
  selfsubjectrulesreviews.authorization.openshift.io  []             []               [create]
  clusterroles.rbac.authorization.k8s.io          []              []             [get list watch]
  clusterroles                         []            []             [get list]
  clusterroles.authorization.openshift.io          []              []             [get list]
  storageclasses.storage.k8s.io                []              []             [get list]
  users                          []          [~]         [get]
  users.user.openshift.io                 []             [~]          [get]
  projects                        []           []           [list watch]
  projects.project.openshift.io              []             []              [list watch]
  projectrequests                     []           []            [list]
  projectrequests.project.openshift.io           []             []              [list]


Name:       cluster-admin
Labels:     kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
Resources  Non-Resource URLs  Resource Names  Verbs
---------  ----------------   --------------  -----
*.*      []            []          [*]
         [*]            []          [*]

...
```

2. To view the current set of cluster role bindings, which shows the users and groups that are bound to various roles:

```
$ oc describe clusterrolebinding.rbac
```

Example output

```
Name:       alertmanager-main
Labels:     <none>
Annotations:  <none>
```

```
Role:
  Kind: ClusterRole
  Name: alertmanager-main
Subjects:
  Kind           Name            Namespace
  ----           ----            ---------
  ServiceAccount  alertmanager-main  openshift-monitoring


Name:       basic-users
Labels:       <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind: ClusterRole
  Name: basic-user
Subjects:
  Kind   Name            Namespace
  ----   ----            ---------
  Group  system:authenticated


Name:       cloud-credential-operator-rolebinding
Labels:       <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: cloud-credential-operator-role
Subjects:
  Kind           Name     Namespace
  ----           ----     ---------
  ServiceAccount  default  openshift-cloud-credential-operator


Name:       cluster-admin
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations: rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind: ClusterRole
  Name: cluster-admin
Subjects:
  Kind   Name            Namespace
  ----   ----            ---------
  Group  system:masters


Name:       cluster-admins
Labels:       <none>
Annotations: rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind: ClusterRole
  Name: cluster-admin
Subjects:
  Kind   Name            Namespace
  ----   ----            ---------
  Group  system:cluster-admins
  User   system:admin
```

```
Name:        cluster-api-manager-rolebinding
Labels:      <none>
Annotations: <none>
Role:
 Kind:  ClusterRole
 Name:  cluster-api-manager-role
Subjects:
 Kind           Name     Namespace
 ----           ----     ---------
 ServiceAccount  default  openshift-machine-api


...
```

10.2.5. Viewing local roles and bindings

You can use the **oc** CLI to view local roles and bindings by using the **oc describe** command.

Prerequisites

- Install the **oc** CLI.

- Obtain permission to view the local roles and bindings:

  - Users with the **cluster-admin** default cluster role bound cluster-wide can perform any action on any resource, including viewing local roles and bindings.

  - Users with the **admin** default cluster role bound locally can view and manage roles and bindings in that project.

Procedure

1. To view the current set of local role bindings, which show the users and groups that are bound to various roles for the current project:

   ```
   $ oc describe rolebinding.rbac
   ```

2. To view the local role bindings for a different project, add the **-n** flag to the command:

   ```
   $ oc describe rolebinding.rbac -n joe-project
   ```

   Example output

   ```
   Name:        admin
   Labels:      <none>
   Annotations: <none>
   Role:
    Kind:  ClusterRole
    Name:  admin
   Subjects:
    Kind  Name       Namespace
    ----  ----       ---------
    User  kube:admin
   ```

```
Name:       system:deployers
Labels:     <none>
Annotations:  openshift.io/description:
              Allows deploymentconfigs in this namespace to rollout pods in
              this namespace.  It is auto-managed by a controller; remove
              subjects to disa...
Role:
 Kind:  ClusterRole
 Name:  system:deployer
Subjects:
 Kind           Name      Namespace
 ----           ----      ---------
 ServiceAccount  deployer  joe-project


Name:       system:image-builders
Labels:     <none>
Annotations:  openshift.io/description:
              Allows builds in this namespace to push images to this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
 Kind:  ClusterRole
 Name:  system:image-builder
Subjects:
 Kind           Name     Namespace
 ----           ----     ---------
 ServiceAccount  builder  joe-project


Name:       system:image-pullers
Labels:     <none>
Annotations:  openshift.io/description:
              Allows all pods in this namespace to pull images from this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
 Kind:  ClusterRole
 Name:  system:image-puller
Subjects:
 Kind   Name                            Namespace
 ----   ----                            ---------
 Group  system:serviceaccounts:joe-project
```

10.2.6. Adding roles to users

You can use the **oc adm** administrator CLI to manage the roles and bindings.

Binding, or adding, a role to users or groups gives the user or group the access that is granted by the role. You can add and remove roles to and from users and groups using **oc adm policy** commands.

You can bind any of the default cluster roles to local users or groups in your project.

Procedure

1. Add a role to a user in a specific project:

   ```
   $ oc adm policy add-role-to-user <role> <user> -n <project>
   ```

   For example, you can add the **admin** role to the **alice** user in **joe** project by running:

   ```
   $ oc adm policy add-role-to-user admin alice -n joe
   ```

   TIP

   You can alternatively apply the following YAML to add the role to the user:

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: RoleBinding
   metadata:
     name: admin-0
     namespace: joe
   roleRef:
     apiGroup: rbac.authorization.k8s.io
     kind: ClusterRole
     name: admin
   subjects:
   - apiGroup: rbac.authorization.k8s.io
     kind: User
     name: alice
   ```

2. View the local role bindings and verify the addition in the output:

   ```
   $ oc describe rolebinding.rbac -n <project>
   ```

   For example, to view the local role bindings for the **joe** project:

   ```
   $ oc describe rolebinding.rbac -n joe
   ```

   Example output

   ```
   Name:        admin
   Labels:      <none>
   Annotations: <none>
   Role:
     Kind:  ClusterRole
     Name:  admin
   Subjects:
     Kind  Name       Namespace
     ----  ----       ---------
     User  kube:admin


   Name:        admin-0
   Labels:      <none>
   Annotations: <none>
   ```

```
Role:
  Kind:  ClusterRole
  Name:  admin
Subjects:
  Kind  Name   Namespace
  ----  ----   ---------
  User  alice
```
 1

```
Name:        system:deployers
Labels:       <none>
Annotations:  openshift.io/description:
              Allows deploymentconfigs in this namespace to rollout pods in
              this namespace.  It is auto-managed by a controller; remove
              subjects to disa...
Role:
  Kind:  ClusterRole
  Name:  system:deployer
Subjects:
  Kind            Name      Namespace
  ----            ----      ---------
  ServiceAccount  deployer  joe
```

```
Name:        system:image-builders
Labels:       <none>
Annotations:  openshift.io/description:
              Allows builds in this namespace to push images to this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
  Kind:  ClusterRole
  Name:  system:image-builder
Subjects:
  Kind            Name     Namespace
  ----            ----     ---------
  ServiceAccount  builder  joe
```

```
Name:        system:image-pullers
Labels:       <none>
Annotations:  openshift.io/description:
              Allows all pods in this namespace to pull images from this
              namespace.  It is auto-managed by a controller; remove subjects
              to disable.
Role:
  Kind:  ClusterRole
  Name:  system:image-puller
Subjects:
  Kind   Name                    Namespace
  ----   ----                    ---------
  Group  system:serviceaccounts:joe
```

1 The **alice** user has been added to the **admins RoleBinding**.

10.2.7. Creating a local role

You can create a local role for a project and then bind it to a user.

Procedure

1. To create a local role for a project, run the following command:

```
$ oc create role <name> --verb=<verb> --resource=<resource> -n <project>
```

In this command, specify:

- **<name>**, the local role's name

- **<verb>**, a comma-separated list of the verbs to apply to the role

- **<resource>**, the resources that the role applies to

- **<project>**, the project name

For example, to create a local role that allows a user to view pods in the **blue** project, run the following command:

```
$ oc create role podview --verb=get --resource=pod -n blue
```

2. To bind the new role to a user, run the following command:

```
$ oc adm policy add-role-to-user podview user2 --role-namespace=blue -n blue
```

10.2.8. Creating a cluster role

You can create a cluster role.

Procedure

1. To create a cluster role, run the following command:

```
$ oc create clusterrole <name> --verb=<verb> --resource=<resource>
```

In this command, specify:

- **<name>**, the local role's name

- **<verb>**, a comma-separated list of the verbs to apply to the role

- **<resource>**, the resources that the role applies to

For example, to create a cluster role that allows a user to view pods, run the following command:

```
$ oc create clusterrole podviewonly --verb=get --resource=pod
```

10.2.9. Local role binding commands

When you manage a user or group's associated roles for local role bindings using the following operations, a project may be specified with the **-n** flag. If it is not specified, then the current project is used.

You can use the following commands for local RBAC management.

Table 10.1. Local role binding operations

Command	Description
$ oc adm policy who-can *<verb> <resource>*	Indicates which users can perform an action on a resource.
$ oc adm policy add-role-to-user *<role> <username>*	Binds a specified role to specified users in the current project.
$ oc adm policy remove-role-from-user *<role> <username>*	Removes a given role from specified users in the current project.
$ oc adm policy remove-user *<username>*	Removes specified users and all of their roles in the current project.
$ oc adm policy add-role-to-group *<role> <groupname>*	Binds a given role to specified groups in the current project.
$ oc adm policy remove-role-from-group *<role> <groupname>*	Removes a given role from specified groups in the current project.
$ oc adm policy remove-group *<groupname>*	Removes specified groups and all of their roles in the current project.

10.2.10. Cluster role binding commands

You can also manage cluster role bindings using the following operations. The **-n** flag is not used for these operations because cluster role bindings use non-namespaced resources.

Table 10.2. Cluster role binding operations

Command	Description
$ oc adm policy add-cluster-role-to-user *<role> <username>*	Binds a given role to specified users for all projects in the cluster.
$ oc adm policy remove-cluster-role-from-user *<role> <username>*	Removes a given role from specified users for all projects in the cluster.
$ oc adm policy add-cluster-role-to-group *<role> <groupname>*	Binds a given role to specified groups for all projects in the cluster.

Command	Description
$ oc adm policy remove-cluster-role-from-group *<role> <groupname>*	Removes a given role from specified groups for all projects in the cluster.

10.2.11. Creating a cluster admin

The **cluster-admin** role is required to perform administrator level tasks on the OpenShift Container Platform cluster, such as modifying cluster resources.

Prerequisites

- You must have created a user to define as the cluster admin.

Procedure

- Define the user as a cluster admin:

      $ oc adm policy add-cluster-role-to-user cluster-admin <user>

10.3. THE KUBEADMIN USER

OpenShift Container Platform creates a cluster administrator, **kubeadmin**, after the installation process completes.

This user has the **cluster-admin** role automatically applied and is treated as the root user for the cluster. The password is dynamically generated and unique to your OpenShift Container Platform environment. After installation completes the password is provided in the installation program's output. For example:

    INFO Install complete!
    INFO Run 'export KUBECONFIG=<your working directory>/auth/kubeconfig' to manage the cluster
    with 'oc', the OpenShift CLI.
    INFO The cluster is ready when 'oc login -u kubeadmin -p <provided>' succeeds (wait a few minutes).
    INFO Access the OpenShift web-console here: https://console-openshift-
    console.apps.demo1.openshift4-beta-abcorp.com
    INFO Login to the console with user: kubeadmin, password: <provided>

10.3.1. Removing the kubeadmin user

After you define an identity provider and create a new **cluster-admin** user, you can remove the **kubeadmin** to improve cluster security.

> **WARNING**
>
> If you follow this procedure before another user is a **cluster-admin**, then OpenShift Container Platform must be reinstalled. It is not possible to undo this command.

Prerequisites

- You must have configured at least one identity provider.

- You must have added the **cluster-admin** role to a user.

- You must be logged in as an administrator.

Procedure

- Remove the **kubeadmin** secrets:

```
$ oc delete secrets kubeadmin -n kube-system
```

10.4. IMAGE CONFIGURATION

Understand and configure image registry settings.

10.4.1. Image controller configuration parameters

The **image.config.openshift.io/cluster** resource holds cluster-wide information about how to handle images. The canonical, and only valid name is **cluster**. Its **spec** offers the following configuration parameters.

> **NOTE**
>
> Parameters such as **DisableScheduledImport**, **MaxImagesBulkImportedPerRepository**, **MaxScheduledImportsPerMinute**, **ScheduledImageImportMinimumIntervalSeconds**, **InternalRegistryHostname** are not configurable.

Parameter	Description
allowedRegistriesForImport	Limits the container image registries from which normal users can import images. Set this list to the registries that you trust to contain valid images, and that you want applications to be able to import from. Users with permission to create images or **ImageStreamMappings** from the API are not affected by this policy. Typically only cluster administrators have the appropriate permissions. Every element of this list contains a location of the registry specified by the registry domain name. **domainName**: Specifies a domain name for the registry. If the registry uses a non-standard **80** or **443** port, the port should be included in the domain name as well. **insecure**: Insecure indicates whether the registry is secure or insecure. By default, if not otherwise specified, the registry is assumed to be secure.

Parameter	Description
additionalTrustedCA	A reference to a config map containing additional CAs that should be trusted during **image stream import**, **pod image pull**, **openshift-image-registry pullthrough**, and builds. The namespace for this config map is **openshift-config**. The format of the config map is to use the registry hostname as the key, and the PEM-encoded certificate as the value, for each additional registry CA to trust.
externalRegistryHostnames	Provides the hostnames for the default external image registry. The external hostname should be set only when the image registry is exposed externally. The first value is used in **publicDockerImageRepository** field in image streams. The value must be in **hostname[:port]** format.
registrySources	Contains configuration that determines how the container runtime should treat individual registries when accessing images for builds and pods. For instance, whether or not to allow insecure access. It does not contain configuration for the internal cluster registry. **insecureRegistries**: Registries which do not have a valid TLS certificate or only support HTTP connections. To specify all subdomains, add the asterisk (*****) wildcard character as a prefix to the domain name. For example, ***.example.com**. You can specify an individual repository within a registry. For example: **reg1.io/myrepo/myapp:latest**. **blockedRegistries**: Registries for which image pull and push actions are denied. To specify all subdomains, add the asterisk (*****) wildcard character as a prefix to the domain name. For example, ***.example.com**. You can specify an individual repository within a registry. For example: **reg1.io/myrepo/myapp:latest**. All other registries are allowed. **allowedRegistries**: Registries for which image pull and push actions are allowed. To specify all subdomains, add the asterisk (*****) wildcard character as a prefix to the domain name. For example, ***.example.com**. You can specify an individual repository within a registry. For example: **reg1.io/myrepo/myapp:latest**. All other registries are blocked. **containerRuntimeSearchRegistries**: Registries for which image pull and push actions are allowed using image short names. All other registries are blocked. Either **blockedRegistries** or **allowedRegistries** can be set, but not both.

> **WARNING**
>
> When the **allowedRegistries** parameter is defined, all registries, including **registry.redhat.io** and **quay.io** registries and the default OpenShift image registry, are blocked unless explicitly listed. When using the parameter, to prevent pod failure, add all registries including the **registry.redhat.io** and **quay.io** registries and the **internalRegistryHostname** to the **allowedRegistries** list, as they are required by payload images within your environment. For disconnected clusters, mirror registries should also be added.

The **status** field of the **image.config.openshift.io/cluster** resource holds observed values from the cluster.

Parameter	Description
internalRegistryHostname	Set by the Image Registry Operator, which controls the **internalRegistryHostname**. It sets the hostname for the default OpenShift image registry. The value must be in **hostname[:port]** format. For backward compatibility, you can still use the **OPENSHIFT_DEFAULT_REGISTRY** environment variable, but this setting overrides the environment variable.
externalRegistryHostnames	Set by the Image Registry Operator, provides the external hostnames for the image registry when it is exposed externally. The first value is used in **publicDockerImageRepository** field in image streams. The values must be in **hostname[:port]** format.

10.4.2. Configuring image registry settings

You can configure image registry settings by editing the **image.config.openshift.io/cluster** custom resource (CR). When changes to the registry are applied to the **image.config.openshift.io/cluster** CR, the Machine Config Operator (MCO) performs the following sequential actions:

1. Cordons the node

2. Applies changes by restarting CRI-O

3. Uncordons the node
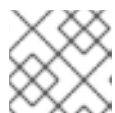
> **NOTE**
>
> The MCO does not restart nodes when it detects changes.

Procedure

1. Edit the **image.config.openshift.io/cluster** custom resource:

   ```
   $ oc edit image.config.openshift.io/cluster
   ```

The following is an example **image.config.openshift.io/cluster** CR:

```
apiVersion: config.openshift.io/v1
kind: Image 1
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  allowedRegistriesForImport: 2
    - domainName: quay.io
      insecure: false
  additionalTrustedCA: 3
    name: myconfigmap
  registrySources: 4
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - image-registry.openshift-image-registry.svc:5000
    - reg1.io/myrepo/myapp:latest
    insecureRegistries:
    - insecure.com
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1   **Image**: Holds cluster-wide information about how to handle images. The canonical, and only valid name is **cluster**.

2   **allowedRegistriesForImport**: Limits the container image registries from which normal users may import images. Set this list to the registries that you trust to contain valid images, and that you want applications to be able to import from. Users with permission to create images or **ImageStreamMappings** from the API are not affected by this policy. Typically only cluster administrators have the appropriate permissions.

3   **additionalTrustedCA**: A reference to a config map containing additional certificate authorities (CA) that are trusted during image stream import, pod image pull, **openshift-image-registry** pullthrough, and builds. The namespace for this config map is **openshift-config**. The format of the config map is to use the registry hostname as the key, and the PEM certificate as the value, for each additional registry CA to trust.

4   **registrySources**: Contains configuration that determines whether the container runtime allows or blocks individual registries when accessing images for builds and pods. Either the **allowedRegistries** parameter or the **blockedRegistries** parameter can be set, but not both. You can also define whether or not to allow access to insecure registries or registries that allow registries that use image short names. This example uses the **allowedRegistries** parameter, which defines the registries that are allowed to be used. The insecure registry **insecure.com** is also allowed. The **registrySources** parameter does not contain configuration for the internal cluster registry.

NOTE

When the **allowedRegistries** parameter is defined, all registries, including the registry.redhat.io and quay.io registries and the default OpenShift image registry, are blocked unless explicitly listed. If you use the parameter, to prevent pod failure, you must add the **registry.redhat.io** and **quay.io** registries and the **internalRegistryHostname** to the **allowedRegistries** list, as they are required by payload images within your environment. Do not add the **registry.redhat.io** and **quay.io** registries to the **blockedRegistries** list.

When using the **allowedRegistries**, **blockedRegistries**, or **insecureRegistries** parameter, you can specify an individual repository within a registry. For example: **reg1.io/myrepo/myapp:latest**.

Insecure external registries should be avoided to reduce possible security risks.

2. To check that the changes are applied, list your nodes:

```
$ oc get nodes
```

Example output

```
NAME                                        STATUS               ROLES         AGE   VERSION
ip-10-0-137-182.us-east-2.compute.internal   Ready,SchedulingDisabled   worker
65m   v1.25.4+77bec7a
ip-10-0-139-120.us-east-2.compute.internal   Ready,SchedulingDisabled   control-plane
74m   v1.25.4+77bec7a
ip-10-0-176-102.us-east-2.compute.internal   Ready                      control-plane       75m
v1.25.4+77bec7a
ip-10-0-188-96.us-east-2.compute.internal    Ready                      worker              65m
v1.25.4+77bec7a
ip-10-0-200-59.us-east-2.compute.internal    Ready                      worker              63m
v1.25.4+77bec7a
ip-10-0-223-123.us-east-2.compute.internal   Ready                      control-plane       73m
v1.25.4+77bec7a
```

For more information on the allowed, blocked, and insecure registry parameters, see Configuring image registry settings.

10.4.2.1. Configuring additional trust stores for image registry access

The **image.config.openshift.io/cluster** custom resource can contain a reference to a config map that contains additional certificate authorities to be trusted during image registry access.

Prerequisites

- The certificate authorities (CA) must be PEM-encoded.

Procedure

You can create a config map in the **openshift-config** namespace and use its name in **AdditionalTrustedCA** in the **image.config.openshift.io** custom resource to provide additional CAs that should be trusted when contacting external registries.

The config map key is the hostname of a registry with the port for which this CA is to be trusted, and the PEM certificate content is the value, for each additional registry CA to trust.

Image registry CA config map example

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-registry-ca
data:
  registry.example.com: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
  registry-with-port.example.com..5000: | 1
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

1 If the registry has the port, such as **registry-with-port.example.com:5000**, **:** should be replaced with **..**.

You can configure additional CAs with the following procedure.

1. To configure an additional CA:

   ```
   $ oc create configmap registry-config --from-file=<external_registry_address>=ca.crt -n openshift-config
   ```

   ```
   $ oc edit image.config.openshift.io cluster
   ```

   ```
   spec:
     additionalTrustedCA:
       name: registry-config
   ```

10.4.2.2. Configuring image registry repository mirroring

Setting up container registry repository mirroring enables you to do the following:

- Configure your OpenShift Container Platform cluster to redirect requests to pull images from a repository on a source image registry and have it resolved by a repository on a mirrored image registry.

- Identify multiple mirrored repositories for each target repository, to make sure that if one mirror is down, another can be used.

The attributes of repository mirroring in OpenShift Container Platform include:

- Image pulls are resilient to registry downtimes.

- Clusters in disconnected environments can pull images from critical locations, such as quay.io, and have registries behind a company firewall provide the requested images.

- A particular order of registries is tried when an image pull request is made, with the permanent registry typically being the last one tried.

- The mirror information you enter is added to the **/etc/containers/registries.conf** file on every node in the OpenShift Container Platform cluster.

- When a node makes a request for an image from the source repository, it tries each mirrored repository in turn until it finds the requested content. If all mirrors fail, the cluster tries the source repository. If successful, the image is pulled to the node.

Setting up repository mirroring can be done in the following ways:

- At OpenShift Container Platform installation:
  By pulling container images needed by OpenShift Container Platform and then bringing those images behind your company's firewall, you can install OpenShift Container Platform into a datacenter that is in a disconnected environment.

- After OpenShift Container Platform installation:
  Even if you don't configure mirroring during OpenShift Container Platform installation, you can do so later using the **ImageContentSourcePolicy** object.

The following procedure provides a postinstallation mirror configuration, where you create an **ImageContentSourcePolicy** object that identifies:

- The source of the container image repository you want to mirror.

- A separate entry for each mirror repository you want to offer the content requested from the source repository.

> **NOTE**
>
> You can only configure global pull secrets for clusters that have an **ImageContentSourcePolicy** object. You cannot add a pull secret to a project.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Configure mirrored repositories, by either:

   - Setting up a mirrored repository with Red Hat Quay, as described in Red Hat Quay Repository Mirroring. Using Red Hat Quay allows you to copy images from one repository to another and also automatically sync those repositories repeatedly over time.

   - Using a tool such as **skopeo** to copy images manually from the source directory to the mirrored repository.
     For example, after installing the skopeo RPM package on a Red Hat Enterprise Linux (RHEL) 7 or RHEL 8 system, use the **skopeo** command as shown in this example:

     ```
     $ skopeo copy \
     docker://registry.access.redhat.com/ubi8/ubi-
     minimal@sha256:5cfbaf45ca96806917830c183e9f37df2e913b187adb32e89fd83fa455eba
     a6 \
     docker://example.io/example/ubi-minimal
     ```

In this example, you have a container image registry that is named **example.io** with an image repository named **example** to which you want to copy the **ubi8/ubi-minimal** image from **registry.access.redhat.com**. After you create the registry, you can configure your OpenShift Container Platform cluster to redirect requests made of the source repository to the mirrored repository.

2. Log in to your OpenShift Container Platform cluster.

3. Create an **ImageContentSourcePolicy** file (for example, **registryrepomirror.yaml**), replacing the source and mirrors with your own registry and repository pairs and images:

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: ubi8repo
spec:
 repositoryDigestMirrors:
 - mirrors:
   - example.io/example/ubi-minimal 1
   - example.com/example/ubi-minimal 2
   source: registry.access.redhat.com/ubi8/ubi-minimal 3
 - mirrors:
   - mirror.example.com/redhat
   source: registry.redhat.io/openshift4 4
 - mirrors:
   - mirror.example.com
   source: registry.redhat.io 5
 - mirrors:
   - mirror.example.net/image
   source: registry.example.com/example/myimage 6
 - mirrors:
   - mirror.example.net
   source: registry.example.com/example 7
 - mirrors:
   - mirror.example.net/registry-example-com
   source: registry.example.com 8
```

[1] Indicates the name of the image registry and repository.

[2] Indicates multiple mirror repositories for each target repository. If one mirror is down, the target repository can use another mirror.

[3] Indicates the registry and repository containing the content that is mirrored.

[4] You can configure a namespace inside a registry to use any image in that namespace. If you use a registry domain as a source, the **ImageContentSourcePolicy** resource is applied to all repositories from the registry.

[5] If you configure the registry name, the **ImageContentSourcePolicy** resource is applied to all repositories from a source registry to a mirror registry.

[6] Pulls the image **mirror.example.net/image@sha256:…**.

[7] Pulls the image **myimage** in the source registry namespace from the mirror **mirror.example.net/myimage@sha256:…**.

8 Pulls the image **registry.example.com/example/myimage** from the mirror registry **mirror.example.net/registry-example-com/example/myimage@sha256:…**. The

4. Create the new **ImageContentSourcePolicy** object:

```
$ oc create -f registryrepomirror.yaml
```

After the **ImageContentSourcePolicy** object is created, the new settings are deployed to each node and the cluster starts using the mirrored repository for requests to the source repository.

5. To check that the mirrored configuration settings, are applied, do the following on one of the nodes.

   a. List your nodes:

   ```
   $ oc get node
   ```

   Example output

   ```
   NAME                         STATUS       ROLES   AGE  VERSION
   ip-10-0-137-44.ec2.internal   Ready         worker  7m   v1.25.0
   ip-10-0-138-148.ec2.internal  Ready         master  11m  v1.25.0
   ip-10-0-139-122.ec2.internal  Ready         master  11m  v1.25.0
   ip-10-0-147-35.ec2.internal   Ready         worker  7m   v1.25.0
   ip-10-0-153-12.ec2.internal   Ready         worker  7m   v1.25.0
   ip-10-0-154-10.ec2.internal   Ready         master  11m  v1.25.0
   ```

   The **Imagecontentsourcepolicy** resource does not restart the nodes.

   b. Start the debugging process to access the node:

   ```
   $ oc debug node/ip-10-0-147-35.ec2.internal
   ```

   Example output

   ```
   Starting pod/ip-10-0-147-35ec2internal-debug ...
   To use host binaries, run `chroot /host`
   ```

   c. Change your root directory to **/host**:

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

   d. Check the **/etc/containers/registries.conf** file to make sure the changes were made:

   ```
   sh-4.2# cat /etc/containers/registries.conf
   ```

   Example output

   ```
   unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]
   short-name-mode = ""

   [[registry]]
   ```

```
    prefix = ""
    location = "registry.access.redhat.com/ubi8/ubi-minimal"
    mirror-by-digest-only = true

    [[registry.mirror]]
      location = "example.io/example/ubi-minimal"

    [[registry.mirror]]
      location = "example.com/example/ubi-minimal"

  [[registry]]
    prefix = ""
    location = "registry.example.com"
    mirror-by-digest-only = true

    [[registry.mirror]]
      location = "mirror.example.net/registry-example-com"

  [[registry]]
    prefix = ""
    location = "registry.example.com/example"
    mirror-by-digest-only = true

    [[registry.mirror]]
      location = "mirror.example.net"

  [[registry]]
    prefix = ""
    location = "registry.example.com/example/myimage"
    mirror-by-digest-only = true

    [[registry.mirror]]
      location = "mirror.example.net/image"

  [[registry]]
    prefix = ""
    location = "registry.redhat.io"
    mirror-by-digest-only = true

    [[registry.mirror]]
      location = "mirror.example.com"

  [[registry]]
    prefix = ""
    location = "registry.redhat.io/openshift4"
    mirror-by-digest-only = true

    [[registry.mirror]]
      location = "mirror.example.com/redhat"
```

e. Pull an image digest to the node from the source and check if it is resolved by the mirror.
   ImageContentSourcePolicy objects support image digests only, not image tags.

```
sh-4.2# podman pull --log-level=debug registry.access.redhat.com/ubi8/ubi-
minimal@sha256:5cfbaf45ca96806917830c183e9f37df2e913b187adb32e89fd83fa455eba
a6
```

Troubleshooting repository mirroring

If the repository mirroring procedure does not work as described, use the following information about how repository mirroring works to help troubleshoot the problem.

- The first working mirror is used to supply the pulled image.

- The main registry is only used if no other mirror works.

- From the system context, the **Insecure** flags are used as fallback.

- The format of the **/etc/containers/registries.conf** file has changed recently. It is now version 2 and in TOML format.

10.5. POPULATING OPERATORHUB FROM MIRRORED OPERATOR CATALOGS

If you mirrored Operator catalogs for use with disconnected clusters, you can populate OperatorHub with the Operators from your mirrored catalogs. You can use the generated manifests from the mirroring process to create the required **ImageContentSourcePolicy** and **CatalogSource** objects.

10.5.1. Prerequisites

- Mirroring Operator catalogs for use with disconnected clusters

10.5.2. Creating the ImageContentSourcePolicy object

After mirroring Operator catalog content to your mirror registry, create the required **ImageContentSourcePolicy** (ICSP) object. The ICSP object configures nodes to translate between the image references stored in Operator manifests and the mirrored registry.

Procedure

- On a host with access to the disconnected cluster, create the ICSP by running the following command to specify the **imageContentSourcePolicy.yaml** file in your manifests directory:

  ```
  $ oc create -f <path/to/manifests/dir>/imageContentSourcePolicy.yaml
  ```

  where **<path/to/manifests/dir>** is the path to the manifests directory for your mirrored content.

  You can now create a **CatalogSource** object to reference your mirrored index image and Operator content.

10.5.3. Adding a catalog source to a cluster

Adding a catalog source to an OpenShift Container Platform cluster enables the discovery and installation of Operators for users. Cluster administrators can create a **CatalogSource** object that references an index image. OperatorHub uses catalog sources to populate the user interface.

TIP

Alternatively, you can use the web console to manage catalog sources. From the **Administration →
Cluster Settings → Configuration → OperatorHub** page, click the **Sources** tab, where you can create,
update, delete, disable, and enable individual sources.

Prerequisites

- An index image built and pushed to a registry.

Procedure

1. Create a **CatalogSource** object that references your index image. If you used the **oc adm
catalog mirror** command to mirror your catalog to a target registry, you can use the generated
catalogSource.yaml file in your manifests directory as a starting point.

   a. Modify the following to your specifications and save it as a **catalogSource.yaml** file:

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: CatalogSource
   metadata:
     name: my-operator-catalog 1
     namespace: openshift-marketplace 2
   spec:
     sourceType: grpc
     grpcPodConfig:
       securityContextConfig: <security_mode> 3
     image: <registry>/<namespace>/redhat-operator-index:v4.12 4
     displayName: My Operator Catalog
     publisher: <publisher_name> 5
     updateStrategy:
       registryPoll: 6
         interval: 30m
   ```

   1 If you mirrored content to local files before uploading to a registry, remove any
   backslash (/) characters from the **metadata.name** field to avoid an "invalid resource
   name" error when you create the object.

   2 If you want the catalog source to be available globally to users in all namespaces,
   specify the **openshift-marketplace** namespace. Otherwise, you can specify a different
   namespace for the catalog to be scoped and available only for that namespace.

   3 Specify the value of **legacy** or **restricted**. If the field is not set, the default value is
   legacy. In a future OpenShift Container Platform release, it is planned that the default
   value will be **restricted**. If your catalog cannot run with **restricted** permissions, it is
   recommended that you manually set this field to **legacy**.

   4 Specify your index image. If you specify a tag after the image name, for example
   :v4.12, the catalog source pod uses an image pull policy of **Always**, meaning the pod
   always pulls the image prior to starting the container. If you specify a digest, for
   example **@sha256:<id>**, the image pull policy is **IfNotPresent**, meaning the pod pulls
   the image only if it does not already exist on the node.

   5 Specify your name or an organization name publishing the catalog.

6 Catalog sources can automatically check for new versions to keep up to date.

b. Use the file to create the **CatalogSource** object:

```
$ oc apply -f catalogSource.yaml
```

2. Verify the following resources are created successfully.

a. Check the pods:

```
$ oc get pods -n openshift-marketplace
```

Example output

```
NAME                             READY   STATUS    RESTARTS AGE
my-operator-catalog-6njx6              1/1    Running  0      28s
marketplace-operator-d9f549946-96sgr   1/1    Running  0      26h
```

b. Check the catalog source:

```
$ oc get catalogsource -n openshift-marketplace
```

Example output

```
NAME            DISPLAY          TYPE PUBLISHER  AGE
my-operator-catalog   My Operator Catalog   grpc          5s
```

c. Check the package manifest:

```
$ oc get packagemanifest -n openshift-marketplace
```

Example output

```
NAME                CATALOG          AGE
jaeger-product            My Operator Catalog   93s
```

You can now install the Operators from the **OperatorHub** page on your OpenShift Container Platform web console.

Additional resources

- Accessing images for Operators from private registries
- Image template for custom catalog sources
- Image pull policy

10.6. ABOUT OPERATOR INSTALLATION WITH OPERATORHUB

OperatorHub is a user interface for discovering Operators; it works in conjunction with Operator Lifecycle Manager (OLM), which installs and manages Operators on a cluster.

As a cluster administrator, you can install an Operator from OperatorHub by using the OpenShift Container Platform web console or CLI. Subscribing an Operator to one or more namespaces makes the Operator available to developers on your cluster.

During installation, you must determine the following initial settings for the Operator:

Installation Mode

Choose **All namespaces on the cluster (default)** to have the Operator installed on all namespaces or choose individual namespaces, if available, to only install the Operator on selected namespaces. This example chooses **All namespaces...** to make the Operator available to all users and projects.

Update Channel

If an Operator is available through multiple channels, you can choose which channel you want to subscribe to. For example, to deploy from the **stable** channel, if available, select it from the list.

Approval Strategy

You can choose automatic or manual updates.
If you choose automatic updates for an installed Operator, when a new version of that Operator is available in the selected channel, Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without human intervention.

If you select manual updates, when a newer version of an Operator is available, OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the Operator updated to the new version.

10.6.1. Installing from OperatorHub using the web console

You can install and subscribe to an Operator from OperatorHub by using the OpenShift Container Platform web console.

Prerequisites

- Access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.

Procedure

1. Navigate in the web console to the **Operators → OperatorHub** page.

2. Scroll or type a keyword into the **Filter by keyword** box to find the Operator you want. For example, type **jaeger** to find the Jaeger Operator.
   You can also filter options by **Infrastructure Features**. For example, select **Disconnected** if you want to see Operators that work in disconnected environments, also known as restricted network environments.

3. Select the Operator to display additional information.

   > **NOTE**
   >
   > Choosing a Community Operator warns that Red Hat does not certify Community Operators; you must acknowledge the warning before continuing.

4. Read the information about the Operator and click **Install**.

5. On the **Install Operator** page:

   a. Select one of the following:

      - **All namespaces on the cluster (default)**installs the Operator in the default **openshift-operators** namespace to watch and be made available to all namespaces in the cluster. This option is not always available.

      - **A specific namespace on the cluster**allows you to choose a specific, single namespace in which to install the Operator. The Operator will only watch and be made available for use in this single namespace.

   b. Select an **Update Channel** (if more than one is available).

   c. Select **Automatic** or **Manual** approval strategy, as described earlier.

6. Click **Install** to make the Operator available to the selected namespaces on this OpenShift Container Platform cluster.

   a. If you selected a **Manual** approval strategy, the upgrade status of the subscription remains **Upgrading** until you review and approve the install plan.
      After approving on the **Install Plan** page, the subscription upgrade status moves to **Up to date**.

   b. If you selected an **Automatic** approval strategy, the upgrade status should resolve to **Up to date** without intervention.

7. After the upgrade status of the subscription is **Up to date**, select **Operators → Installed Operators** to verify that the cluster service version (CSV) of the installed Operator eventually shows up. The **Status** should ultimately resolve to **InstallSucceeded** in the relevant namespace.

> **NOTE**
>
> For the **All namespaces...** installation mode, the status resolves to **InstallSucceeded** in the **openshift-operators** namespace, but the status is **Copied** if you check in other namespaces.

If it does not:

   a. Check the logs in any pods in the **openshift-operators** project (or other relevant namespace if **A specific namespace...** installation mode was selected) on the **Workloads → Pods** page that are reporting issues to troubleshoot further.

10.6.2. Installing from OperatorHub using the CLI

Instead of using the OpenShift Container Platform web console, you can install an Operator from OperatorHub by using the CLI. Use the **oc** command to create or update a **Subscription** object.

Prerequisites

- Access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.

- Install the **oc** command to your local system.

Procedure

1. View the list of Operators available to the cluster from OperatorHub:

   ```
   $ oc get packagemanifests -n openshift-marketplace
   ```

   Example output

   ```
   NAME                            CATALOG              AGE
   3scale-operator                 Red Hat Operators    91m
   advanced-cluster-management     Red Hat Operators    91m
   amq7-cert-manager               Red Hat Operators    91m
   ...
   couchbase-enterprise-certified  Certified Operators  91m
   crunchy-postgres-operator       Certified Operators  91m
   mongodb-enterprise              Certified Operators  91m
   ...
   etcd                            Community Operators  91m
   jaeger                          Community Operators  91m
   kubefed                         Community Operators  91m
   ...
   ```

   Note the catalog for your desired Operator.

2. Inspect your desired Operator to verify its supported install modes and available channels:

   ```
   $ oc describe packagemanifests <operator_name> -n openshift-marketplace
   ```

3. An Operator group, defined by an **OperatorGroup** object, selects target namespaces in which to generate required RBAC access for all Operators in the same namespace as the Operator group.
   The namespace to which you subscribe the Operator must have an Operator group that matches the install mode of the Operator, either the **AllNamespaces** or **SingleNamespace** mode. If the Operator you intend to install uses the **AllNamespaces** mode, the **openshift-operators** namespace already has the appropriate **global-operators** Operator group in place.

   However, if the Operator uses the **SingleNamespace** mode and you do not already have an appropriate Operator group in place, you must create one.
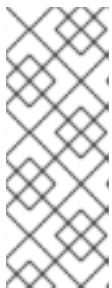
   > **NOTE**
   >
   > - The web console version of this procedure handles the creation of the **OperatorGroup** and **Subscription** objects automatically behind the scenes for you when choosing **SingleNamespace** mode.
   >
   > - You can only have one Operator group per namespace. For more information, see "Operator groups".

   a. Create an **OperatorGroup** object YAML file, for example **operatorgroup.yaml**:

   Example OperatorGroup object

   ```
   apiVersion: operators.coreos.com/v1
   kind: OperatorGroup
   ```

```
  metadata:
    name: <operatorgroup_name>
    namespace: <namespace>
  spec:
    targetNamespaces:
    - <namespace>
```

> **WARNING**
>
> Operator Lifecycle Manager (OLM) creates the following cluster roles for each Operator group:
>
> - **<operatorgroup_name>-admin**
>
> - **<operatorgroup_name>-edit**
>
> - **<operatorgroup_name>-view**
>
> When you manually create an Operator group, you must specify a unique name that does not conflict with the existing cluster roles or other Operator groups on the cluster.

   b. Create the **OperatorGroup** object:

```
$ oc apply -f operatorgroup.yaml
```

4. Create a **Subscription** object YAML file to subscribe a namespace to an Operator, for example **sub.yaml**:

   Example **Subscription object**

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: <subscription_name>
  namespace: openshift-operators 1
spec:
  channel: <channel_name> 2
  name: <operator_name> 3
  source: redhat-operators 4
  sourceNamespace: openshift-marketplace 5
  config:
    env: 6
    - name: ARGS
      value: "-v=10"
    envFrom: 7
    - secretRef:
        name: license-secret
    volumes: 8
```

```
    - name: <volume_name>
      configMap:
        name: <configmap_name>
    volumeMounts: 9
    - mountPath: <directory_name>
      name: <volume_name>
    tolerations: 10
    - operator: "Exists"
    resources: 11
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
    nodeSelector: 12
      foo: bar
```

1 For default **AllNamespaces** install mode usage, specify the **openshift-operators** namespace. Alternatively, you can specify a custom global namespace, if you have created one. Otherwise, specify the relevant single namespace for **SingleNamespace** install mode usage.

2 Name of the channel to subscribe to.

3 Name of the Operator to subscribe to.

4 Name of the catalog source that provides the Operator.

5 Namespace of the catalog source. Use **openshift-marketplace** for the default OperatorHub catalog sources.

6 The **env** parameter defines a list of Environment Variables that must exist in all containers in the pod created by OLM.

7 The **envFrom** parameter defines a list of sources to populate Environment Variables in the container.

8 The **volumes** parameter defines a list of Volumes that must exist on the pod created by OLM.

9 The **volumeMounts** parameter defines a list of VolumeMounts that must exist in all containers in the pod created by OLM. If a **volumeMount** references a **volume** that does not exist, OLM fails to deploy the Operator.

10 The **tolerations** parameter defines a list of Tolerations for the pod created by OLM.

11 The **resources** parameter defines resource constraints for all the containers in the pod created by OLM.

12 The **nodeSelector** parameter defines a **NodeSelector** for the pod created by OLM.

5. Create the **Subscription** object:

```
$ oc apply -f sub.yaml
```

At this point, OLM is now aware of the selected Operator. A cluster service version (CSV) for the Operator should appear in the target namespace, and APIs provided by the Operator should be available for creation.

Additional resources

- [About OperatorGroups](#)

CHAPTER 11. CONFIGURING ALERT NOTIFICATIONS

In OpenShift Container Platform, an alert is fired when the conditions defined in an alerting rule are true. An alert provides a notification that a set of circumstances are apparent within a cluster. Firing alerts can be viewed in the Alerting UI in the OpenShift Container Platform web console by default. After an installation, you can configure OpenShift Container Platform to send alert notifications to external systems.

11.1. SENDING NOTIFICATIONS TO EXTERNAL SYSTEMS

In OpenShift Container Platform 4.12, firing alerts can be viewed in the Alerting UI. Alerts are not configured by default to be sent to any notification systems. You can configure OpenShift Container Platform to send alerts to the following receiver types:

- PagerDuty

- Webhook

- Email

- Slack

Routing alerts to receivers enables you to send timely notifications to the appropriate teams when failures occur. For example, critical alerts require immediate attention and are typically paged to an individual or a critical response team. Alerts that provide non-critical warning notifications might instead be routed to a ticketing system for non-immediate review.

Checking that alerting is operational by using the watchdog alert

OpenShift Container Platform monitoring includes a watchdog alert that fires continuously. Alertmanager repeatedly sends watchdog alert notifications to configured notification providers. The provider is usually configured to notify an administrator when it stops receiving the watchdog alert. This mechanism helps you quickly identify any communication issues between Alertmanager and the notification provider.

11.1.1. Configuring alert receivers

You can configure alert receivers to ensure that you learn about important issues with your cluster.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.

Procedure

1. In the **Administrator** perspective, go to **Administration → Cluster Settings → Configuration → Alertmanager**.

   > **NOTE**
   >
   > Alternatively, you can go to the same page through the notification drawer. Select the bell icon at the top right of the OpenShift Container Platform web console and choose **Configure** in the **AlertmanagerReceiverNotConfigured** alert.

2. Click **Create Receiver** in the **Receivers** section of the page.

3. In the **Create Receiver** form, add a **Receiver name** and choose a **Receiver type** from the list.

4. Edit the receiver configuration:

   - For PagerDuty receivers:

     a. Choose an integration type and add a PagerDuty integration key.

     b. Add the URL of your PagerDuty installation.

     c. Click **Show advanced configuration** if you want to edit the client and incident details or the severity specification.

   - For webhook receivers:

     a. Add the endpoint to send HTTP POST requests to.

     b. Click **Show advanced configuration** if you want to edit the default option to send resolved alerts to the receiver.

   - For email receivers:

     a. Add the email address to send notifications to.

     b. Add SMTP configuration details, including the address to send notifications from, the smarthost and port number used for sending emails, the hostname of the SMTP server, and authentication details.

     c. Select whether TLS is required.

     d. Click **Show advanced configuration** if you want to edit the default option not to send resolved alerts to the receiver or edit the body of email notifications configuration.

   - For Slack receivers:

     a. Add the URL of the Slack webhook.

     b. Add the Slack channel or user name to send notifications to.

     c. Select **Show advanced configuration** if you want to edit the default option not to send resolved alerts to the receiver or edit the icon and username configuration. You can also choose whether to find and link channel names and usernames.

5. By default, firing alerts with labels that match all of the selectors are sent to the receiver. If you want label values for firing alerts to be matched exactly before they are sent to the receiver, perform the following steps:

   a. Add routing label names and values in the **Routing labels** section of the form.

   b. Select **Regular expression** if want to use a regular expression.

   c. Click **Add label** to add further routing labels.

6. Click **Create** to create the receiver.

11.2. ADDITIONAL RESOURCES

- Monitoring overview

- Managing alerts

CHAPTER 12. CONVERTING A CONNECTED CLUSTER TO A DISCONNECTED CLUSTER

There might be some scenarios where you need to convert your OpenShift Container Platform cluster from a connected cluster to a disconnected cluster.

A disconnected cluster, also known as a restricted cluster, does not have an active connection to the internet. As such, you must mirror the contents of your registries and installation media. You can create this mirror registry on a host that can access both the internet and your closed network, or copy images to a device that you can move across network boundaries.

This topic describes the general process for converting an existing, connected cluster into a disconnected cluster.

12.1. ABOUT THE MIRROR REGISTRY

You can mirror the images that are required for OpenShift Container Platform installation and subsequent product updates to a container mirror registry such as Red Hat Quay, JFrog Artifactory, Sonatype Nexus Repository, or Harbor. If you do not have access to a large-scale container registry, you can use the *mirror registry for Red Hat OpenShift*, a small-scale container registry included with OpenShift Container Platform subscriptions.

You can use any container registry that supports Docker v2-2, such as Red Hat Quay, the *mirror registry for Red Hat OpenShift*, Artifactory, Sonatype Nexus Repository, or Harbor. Regardless of your chosen registry, the procedure to mirror content from Red Hat hosted sites on the internet to an isolated image registry is the same. After you mirror the content, you configure each cluster to retrieve this content from your mirror registry.

> **IMPORTANT**
>
> The OpenShift image registry cannot be used as the target registry because it does not support pushing without a tag, which is required during the mirroring process.

If choosing a container registry that is not the *mirror registry for Red Hat OpenShift*, it must be reachable by every machine in the clusters that you provision. If the registry is unreachable, installation, updating, or normal operations such as workload relocation might fail. For that reason, you must run mirror registries in a highly available way, and the mirror registries must at least match the production availability of your OpenShift Container Platform clusters.

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

For mirrored registries, to view the source of pulled images, you must review the **Trying to access** log entry in the CRI-O logs. Other methods to view the image pull source, such as using the **crictl images** command on a node, show the non-mirrored image name, even though the image is pulled from the mirrored location.

> **NOTE**
>
> Red Hat does not test third party registries with OpenShift Container Platform.

12.2. PREREQUISITES

- The **oc** client is installed.

- A running cluster.

- An installed mirror registry, which is a container image registry that supports Docker v2-2 in the location that will host the OpenShift Container Platform cluster, such as one of the following registries:

    - Red Hat Quay

    - JFrog Artifactory

    - Sonatype Nexus Repository

    - Harbor

    If you have an subscription to Red Hat Quay, see the documentation on deploying Red Hat Quay for proof-of-concept purposes or by using the Quay Operator.

- The mirror repository must be configured to share images. For example, a Red Hat Quay repository requires Organizations in order to share images.

- Access to the internet to obtain the necessary container images.

12.3. PREPARING THE CLUSTER FOR MIRRORING

Before disconnecting your cluster, you must mirror, or copy, the images to a mirror registry that is reachable by every node in your disconnected cluster. In order to mirror the images, you must prepare your cluster by:

- Adding the mirror registry certificates to the list of trusted CAs on your host.

- Creating a **.dockerconfigjson** file that contains your image pull secret, which is from the **cloud.openshift.com** token.

Procedure

1. Configuring credentials that allow image mirroring:

    a. Add the CA certificate for the mirror registry, in the simple PEM or DER file formats, to the list of trusted CAs. For example:

    ```
    $ cp </path/to/cert.crt> /usr/share/pki/ca-trust-source/anchors/
    ```

    where, **</path/to/cert.crt>**
    Specifies the path to the certificate on your local file system.

    b. Update the CA trust. For example, in Linux:

    ```
    $ update-ca-trust
    ```

    c. Extract the **.dockerconfigjson** file from the global pull secret:

```
$ oc extract secret/pull-secret -n openshift-config --confirm --to=.
```

Example output

```
.dockerconfigjson
```

d. Edit the **.dockerconfigjson** file to add your mirror registry and authentication credentials and save it as a new file:

```
{"auths":{"<local_registry>": {"auth": "<credentials>","email": "you@example.com"}}},"
<registry>:<port>/<namespace>/":{"auth":"<token>"}}}
```

where:

<local_registry>

Specifies the registry domain name, and optionally the port, that your mirror registry uses to serve content.

auth

Specifies the base64-encoded user name and password for your mirror registry.

<registry>:<port>/<namespace>

Specifies the mirror registry details.

<token>

Specifies the base64-encoded **username:password** for your mirror registry.
For example:

```
$ {"auths":{"cloud.openshift.com":
{"auth":"b3BlbnNoaWZ0Y3UjhGOVZPT0lOMEFaUjdPUzRGTA==","email":"user@exa
mple.com"},
"quay.io":
{"auth":"b3BlbnNoaWZ0LXJlbGVhc2UtZGOVZPT0lOMEFaUGSTd4VGVGVUjdPUzR
GTA==","email":"user@example.com"},
"registry.connect.redhat.com"
{"auth":"NTE3MTMwNDB8dWhjLTFEZlN3VHkxOSTd4VGVGVU1MdTpleUpoYkdjaUail
A==","email":"user@example.com"},
"registry.redhat.io":
{"auth":"NTE3MTMwNDB8dWhjLTFEZlN3VH3BGSTd4VGVGVU1MdTpleUpoYkdjaU9
fZw==","email":"user@example.com"},
"registry.svc.ci.openshift.org":
{"auth":"dXNlcjpyWjAwWVFjSEJiT2RKVW1pSmg4dW92dGp1SXRxQ3RGN1pwajJhN1
ZXeTRV"},"my-registry:5000/my-namespace/":
{"auth":"dXNlcm5hbWU6cGFzc3dvcmQ="}}}
```

12.4. MIRRORING THE IMAGES

After the cluster is properly configured, you can mirror the images from your external repositories to the mirror repository.

Procedure

1. Mirror the Operator Lifecycle Manager (OLM) images:

```
$ oc adm catalog mirror registry.redhat.io/redhat/redhat-operator-index:v{product-version}
<mirror_registry>:<port>/olm -a <reg_creds>
```

where:

product-version

Specifies the tag that corresponds to the version of OpenShift Container Platform to install, such as **4.8**.

mirror_registry

Specifies the fully qualified domain name (FQDN) for the target registry and namespace to mirror the Operator content to, where **<namespace>** is any existing namespace on the registry.

reg_creds

Specifies the location of your modified **.dockerconfigjson** file.

For example:

```
$ oc adm catalog mirror registry.redhat.io/redhat/redhat-operator-index:v4.8
mirror.registry.com:443/olm -a ./.dockerconfigjson  --index-filter-by-os='.*'
```

2. Mirror the content for any other Red Hat-provided Operator:

```
$ oc adm catalog mirror <index_image> <mirror_registry>:<port>/<namespace> -a
<reg_creds>
```

where:

index_image

Specifies the index image for the catalog that you want to mirror.

mirror_registry

Specifies the FQDN for the target registry and namespace to mirror the Operator content to, where **<namespace>** is any existing namespace on the registry.

reg_creds

Optional: Specifies the location of your registry credentials file, if required.

For example:

```
$ oc adm catalog mirror registry.redhat.io/redhat/community-operator-index:v4.8
mirror.registry.com:443/olm -a ./.dockerconfigjson  --index-filter-by-os='.*'
```

3. Mirror the OpenShift Container Platform image repository:

```
$ oc adm release mirror -a .dockerconfigjson --from=quay.io/openshift-release-dev/ocp-
release:v<product-version>-<architecture> --to=<local_registry>/<local_repository> --to-
release-image=<local_registry>/<local_repository>:v<product-version>-<architecture>
```

where:

product-version

Specifies the tag that corresponds to the version of OpenShift Container Platform to install, such as **4.8.15-x86_64**.

architecture

Specifies the type of architecture for your server, such as **x86_64**.

local_registry

Specifies the registry domain name for your mirror repository.

local_repository

Specifies the name of the repository to create in your registry, such as **ocp4/openshift4**.

For example:

```
$ oc adm release mirror -a .dockerconfigjson --from=quay.io/openshift-release-dev/ocp-
release:4.8.15-x86_64 --to=mirror.registry.com:443/ocp/release --to-release-
image=mirror.registry.com:443/ocp/release:4.8.15-x86_64
```

Example output

```
info: Mirroring 109 images to mirror.registry.com/ocp/release ...
mirror.registry.com:443/
  ocp/release
 manifests:
    sha256:086224cadce475029065a0efc5244923f43fb9bb3bb47637e0aaf1f32b9cad47 ->
4.8.15-x86_64-thanos
    sha256:0a214f12737cb1cfbec473cc301aa2c289d4837224c9603e99d1e90fc00328db ->
4.8.15-x86_64-kuryr-controller
    sha256:0cf5fd36ac4b95f9de506623b902118a90ff17a07b663aad5d57c425ca44038c ->
4.8.15-x86_64-pod
    sha256:0d1c356c26d6e5945a488ab2b050b75a8b838fc948a75c0fa13a9084974680cb ->
4.8.15-x86_64-kube-client-agent

…..
sha256:66e37d2532607e6c91eedf23b9600b4db904ce68e92b43c43d5b417ca6c8e63c
mirror.registry.com:443/ocp/release:4.5.41-multus-admission-controller
sha256:d36efdbf8d5b2cbc4dcdbd64297107d88a31ef6b0ec4a39695915c10db4973f1
mirror.registry.com:443/ocp/release:4.5.41-cluster-kube-scheduler-operator
sha256:bd1baa5c8239b23ecdf76819ddb63cd1cd6091119fecdbf1a0db1fb3760321a2
mirror.registry.com:443/ocp/release:4.5.41-aws-machine-controllers
info: Mirroring completed in 2.02s (0B/s)

Success
Update image:  mirror.registry.com:443/ocp/release:4.5.41-x86_64
Mirror prefix: mirror.registry.com:443/ocp/release
```

4. Mirror any other registries, as needed:

```
$ oc image mirror <online_registry>/my/image:latest <mirror_registry>
```

Additional information

- For more information about mirroring Operator catalogs, see Mirroring an Operator catalog .

- For more information about the **oc adm catalog mirror** command, see the OpenShift CLI administrator command reference.

12.5. CONFIGURING THE CLUSTER FOR THE MIRROR REGISTRY

After creating and mirroring the images to the mirror registry, you must modify your cluster so that pods can pull images from the mirror registry.

You must:

- Add the mirror registry credentials to the global pull secret.

- Add the mirror registry server certificate to the cluster.

- Create an **ImageContentSourcePolicy** custom resource (ICSP), which associates the mirror registry with the source registry.

  1. Add mirror registry credential to the cluster global pull-secret:

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

     1   Provide the path to the new pull secret file.

     For example:

     ```
     $ oc set data secret/pull-secret -n openshift-config --from-
     file=.dockerconfigjson=.mirrorsecretconfigjson
     ```

  2. Add the CA-signed mirror registry server certificate to the nodes in the cluster:

     a. Create a config map that includes the server certificate for the mirror registry

        ```
        $ oc create configmap <config_map_name> --from-file=<mirror_address_host>..
        <port>=$path/ca.crt -n openshift-config
        ```

        For example:

        ```
        S oc create configmap registry-config --from-
        file=mirror.registry.com..443=/root/certs/ca-chain.cert.pem -n openshift-config
        ```

     b. Use the config map to update the **image.config.openshift.io/cluster** custom resource (CR). OpenShift Container Platform applies the changes to this CR to all nodes in the cluster:

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

        For example:

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

3. Create an ICSP to redirect container pull requests from the online registries to the mirror registry:

    a. Create the **ImageContentSourcePolicy** custom resource:

    ```
    apiVersion: operator.openshift.io/v1alpha1
    kind: ImageContentSourcePolicy
    metadata:
      name: mirror-ocp
    spec:
      repositoryDigestMirrors:
      - mirrors:
        - mirror.registry.com:443/ocp/release ❶
        source: quay.io/openshift-release-dev/ocp-release ❷
      - mirrors:
        - mirror.registry.com:443/ocp/release
        source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
    ```

    ❶ Specifies the name of the mirror image registry and repository.

    ❷ Specifies the online registry and repository containing the content that is mirrored.

    b. Create the ICSP object:

    ```
    $ oc create -f registryrepomirror.yaml
    ```

    Example output

    ```
    imagecontentsourcepolicy.operator.openshift.io/mirror-ocp created
    ```

    OpenShift Container Platform applies the changes to this CR to all nodes in the cluster.

4. Verify that the credentials, CA, and ICSP for mirror registry were added:

    a. Log into a node:

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

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

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

    c. Check the **config.json** file for the credentials:

    ```
    sh-4.4# cat /var/lib/kubelet/config.json
    ```

    Example output

    ```
    {"auths":{"brew.registry.redhat.io":{"xx=="},"brewregistry.stage.redhat.io":
    {"auth":"xxx=="},"mirror.registry.com:443":{"auth":"xx="}}} ❶
    ```

    ❶ Ensure that the mirror registry and credentials are present.

d. Change to the **certs.d** directory

```
sh-4.4# cd /etc/docker/certs.d/
```

e. List the certificates in the **certs.d** directory:

```
sh-4.4# ls
```

Example output

```
image-registry.openshift-image-registry.svc.cluster.local:5000
image-registry.openshift-image-registry.svc:5000
mirror.registry.com:443 ❶
```

❶   Ensure that the mirror registry is in the list.

f. Check that the ICSP added the mirror registry to the **registries.conf** file:

```
sh-4.4# cat /etc/containers/registries.conf
```

Example output

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

[[registry]]
  prefix = ""
  location = "quay.io/openshift-release-dev/ocp-release"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.registry.com:443/ocp/release"

[[registry]]
  prefix = ""
  location = "quay.io/openshift-release-dev/ocp-v4.0-art-dev"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.registry.com:443/ocp/release"
```

The **registry.mirror** parameters indicate that the mirror registry is searched before the original registry.

g. Exit the node.

```
sh-4.4# exit
```

12.6. ENSURE APPLICATIONS CONTINUE TO WORK

Before disconnecting the cluster from the network, ensure that your cluster is working as expected and all of your applications are working as expected.

Procedure

Use the following commands to check the status of your cluster:

- Ensure your pods are running:

```
$ oc get pods --all-namespaces
```

Example output

```
NAMESPACE                          NAME                                    READY
STATUS    RESTARTS  AGE
kube-system                        apiserver-watcher-ci-ln-47ltxtb-f76d1-mrffg-master-0
1/1    Running   0        39m
kube-system                        apiserver-watcher-ci-ln-47ltxtb-f76d1-mrffg-master-1
1/1    Running   0        39m
kube-system                        apiserver-watcher-ci-ln-47ltxtb-f76d1-mrffg-master-2
1/1    Running   0        39m
openshift-apiserver-operator            openshift-apiserver-operator-79c7c646fd-5rvr5
1/1    Running   3        45m
openshift-apiserver                   apiserver-b944c4645-q694g                2/2
Running   0        29m
openshift-apiserver                   apiserver-b944c4645-shdxb                2/2
Running   0        31m
openshift-apiserver                   apiserver-b944c4645-x7rf2                2/2
Running   0        33m
 ...
```

- Ensure your nodes are in the READY status:

```
$ oc get nodes
```

Example output

```
NAME                            STATUS  ROLES   AGE  VERSION
ci-ln-47ltxtb-f76d1-mrffg-master-0      Ready   master  42m  v1.25.0
ci-ln-47ltxtb-f76d1-mrffg-master-1      Ready   master  42m  v1.25.0
ci-ln-47ltxtb-f76d1-mrffg-master-2      Ready   master  42m  v1.25.0
ci-ln-47ltxtb-f76d1-mrffg-worker-a-gsxbz Ready   worker  35m  v1.25.0
ci-ln-47ltxtb-f76d1-mrffg-worker-b-5qqdx Ready   worker  35m  v1.25.0
ci-ln-47ltxtb-f76d1-mrffg-worker-c-rjkpq Ready   worker  34m  v1.25.0
```

12.7. DISCONNECT THE CLUSTER FROM THE NETWORK

After mirroring all the required repositories and configuring your cluster to work as a disconnected cluster, you can disconnect the cluster from the network.

> **NOTE**
>
> The Insights Operator is degraded when the cluster loses its Internet connection. You can avoid this problem by temporarily disabling the Insights Operator until you can restore it.

12.8. RESTORING A DEGRADED INSIGHTS OPERATOR

Disconnecting the cluster from the network necessarily causes the cluster to lose the Internet connection. The Insights Operator becomes degraded because it requires access to Red Hat Insights.

This topic describes how to recover from a degraded Insights Operator.

Procedure

1. Edit your **.dockerconfigjson** file to remove the **cloud.openshift.com** entry, for example:

   ```
   "cloud.openshift.com":{"auth":"<hash>","email":"user@example.com"}
   ```

2. Save the file.

3. Update the cluster secret with the edited **.dockerconfigjson** file:

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

4. Verify that the Insights Operator is no longer degraded:

   ```
   $ oc get co insights
   ```

   Example output

   ```
   NAME      VERSION  AVAILABLE  PROGRESSING  DEGRADED  SINCE
   insights  4.5.41   True       False        False     3d
   ```

12.9. RESTORING THE NETWORK

If you want to reconnect a disconnected cluster and pull images from online registries, delete the cluster's ImageContentSourcePolicy (ICSP) objects. Without the ICSP, pull requests to external registries are no longer redirected to the mirror registry.

Procedure

1. View the ICSP objects in your cluster:

   ```
   $ oc get imagecontentsourcepolicy
   ```

   Example output

   ```
   NAME          AGE
   mirror-ocp    6d20h
   ocp4-index-0   6d18h
   qe45-index-0   6d15h
   ```

2. Delete all the ICSP objects you created when disconnecting your cluster:

   ```
   $ oc delete imagecontentsourcepolicy <icsp_name> <icsp_name> <icsp_name>
   ```

   For example:

```
$ oc delete imagecontentsourcepolicy mirror-ocp ocp4-index-0 qe45-index-0
```

Example output

```
imagecontentsourcepolicy.operator.openshift.io "mirror-ocp" deleted
imagecontentsourcepolicy.operator.openshift.io "ocp4-index-0" deleted
imagecontentsourcepolicy.operator.openshift.io "qe45-index-0" deleted
```

3. Wait for all the nodes to restart and return to the READY status and verify that the **registries.conf** file is pointing to the original registries and not the mirror registries:

    a. Log into a node:

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

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

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

    c. Examine the **registries.conf** file:

    ```
    sh-4.4# cat /etc/containers/registries.conf
    ```

    Example output

    ```
    unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]    ❶
    ```

    ❶ The **registry** and **registry.mirror** entries created by the ICSPs you deleted are removed.

CHAPTER 13. ENABLING CLUSTER CAPABILITIES

Cluster administrators can enable cluster capabilities that were disabled prior to installation.

> **NOTE**
>
> Cluster administrators cannot disable a cluster capability after it is enabled.

13.1. VIEWING THE CLUSTER CAPABILITIES

As a cluster administrator, you can view the capabilities by using the **clusterversion** resource status.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- To view the status of the cluster capabilities, run the following command:

```
$ oc get clusterversion version -o jsonpath='{.spec.capabilities}{"\n"}{.status.capabilities}{"\n"}'
```

Example output

```
{"additionalEnabledCapabilities":["openshift-samples"],"baselineCapabilitySet":"None"}
{"enabledCapabilities":["openshift-samples"],"knownCapabilities":
["CSISnapshot","Console","Insights","Storage","baremetal","marketplace","openshift-
samples"]}
```

13.2. ENABLING THE CLUSTER CAPABILITIES BY SETTING BASELINE CAPABILITY SET

As a cluster administrator, you can enable the capabilities by setting **baselineCapabilitySet**.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- To set the **baselineCapabilitySet**, run the following command:

```
$ oc patch clusterversion version --type merge -p '{"spec":{"capabilities":
{"baselineCapabilitySet":"vCurrent"}}}'  ❶
```

❶ For **baselineCapabilitySet** you can specify **vCurrent**, **v4.11**, **v4.12**, or **None**.

The following table describes the **baselineCapabilitySet** values.

Table 13.1. Cluster capabilities **baselineCapabilitySet** values description

Value	Description
vCurrent	Specify this option when you want to automatically add new, default capabilities that are introduced in new releases.
v4.11	Specify this option when you want to enable the default capabilities for OpenShift Container Platform 4.11. By specifying **v4.11**, capabilities that are introduced in newer versions of OpenShift Container Platform are not enabled. The default capabilities in OpenShift Container Platform 4.11 are **baremetal**, **marketplace**, and **openshift-samples**.
v4.12	Specify this option when you want to enable the default capabilities for OpenShift Container Platform 4.12. By specifying **v4.12**, capabilities that are introduced in newer versions of OpenShift Container Platform are not enabled. The default capabilities in OpenShift Container Platform 4.12 are **baremetal**, **marketplace**, **openshift-samples**, **Console**, **Insights**, **Storage** and **CSISnapshot**.
None	Specify when the other sets are too large, and you do not need any capabilities or want to fine-tune via **additionalEnabledCapabilities**.

13.3. ENABLING THE CLUSTER CAPABILITIES BY SETTING ADDITIONAL ENABLED CAPABILITIES

As a cluster administrator, you can enable the cluster capabilities by setting **additionalEnabledCapabilities**.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. View the additional enabled capabilities by running the following command:

   ```
   $ oc get clusterversion version -o jsonpath='{.spec.capabilities.additionalEnabledCapabilities}
   {"\n"}'
   ```

   Example output

   ```
   ["openshift-samples"]
   ```

2. To set the **additionalEnabledCapabilities**, run the following command:

   ```
   $ oc patch clusterversion/version --type merge -p '{"spec":{"capabilities":
   {"additionalEnabledCapabilities":["openshift-samples", "marketplace"]}}}'
   ```

IMPORTANT

It is not possible to disable a capability which is already enabled in a cluster. The cluster version Operator (CVO) continues to reconcile the capability which is already enabled in the cluster.

If you try to disable a capability, then CVO shows the divergent spec:

```
$ oc get clusterversion version -o jsonpath='{.status.conditions[?
(@.type=="ImplicitlyEnabledCapabilities")]}{"\n"}'
```

Example output

```
{"lastTransitionTime":"2022-07-22T03:14:35Z","message":"The following capabilities could not be
disabled: openshift-
samples","reason":"CapabilitiesImplicitlyEnabled","status":"True","type":"ImplicitlyEnabledCapabilities"}
```

NOTE

During the cluster upgrades, it is possible that a given capability could be implicitly enabled. If a resource was already running on the cluster before the upgrade, then any capabilities that is part of the resource will be enabled. For example, during a cluster upgrade, a resource that is already running on the cluster has been changed to be part of the **marketplace** capability by the system. Even if a cluster administrator does not explicitly enabled the **marketplace** capability, it is implicitly enabled by the system.

13.4. ADDITIONAL RESOURCES

- Cluster capabilities

CHAPTER 14. CONFIGURING ADDITIONAL DEVICES IN AN IBM Z OR IBM(R) LINUXONE ENVIRONMENT

After installing OpenShift Container Platform, you can configure additional devices for your cluster in an IBM Z or IBM® LinuxONE environment, which is installed with z/VM. The following devices can be configured:

- Fibre Channel Protocol (FCP) host

- FCP LUN

- DASD

- qeth

You can configure devices by adding udev rules using the Machine Config Operator (MCO) or you can configure devices manually.

> **NOTE**
>
> The procedures described here apply only to z/VM installations. If you have installed your cluster with RHEL KVM on IBM Z or IBM® LinuxONE infrastructure, no additional configuration is needed inside the KVM guest after the devices were added to the KVM guests. However, both in z/VM and RHEL KVM environments the next steps to configure the Local Storage Operator and Kubernetes NMState Operator need to be applied.

Additional resources

- [Post-installation machine configuration tasks](#)

14.1. CONFIGURING ADDITIONAL DEVICES USING THE MACHINE CONFIG OPERATOR (MCO)

Tasks in this section describe how to use features of the Machine Config Operator (MCO) to configure additional devices in an IBM Z or IBM® LinuxONE environment. Configuring devices with the MCO is persistent but only allows specific configurations for compute nodes. MCO does not allow control plane nodes to have different configurations.

Prerequisites

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

- The device must be available to the z/VM guest.

- The device is already attached.

- The device is not included in the **cio_ignore** list, which can be set in the kernel parameters.

- You have created a **MachineConfig** object file with the following YAML:

  ```
  apiVersion: machineconfiguration.openshift.io/v1
  kind: MachineConfigPool
  metadata:
    name: worker0
  ```

```
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,worker0]}
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker0: ""
```

14.1.1. Configuring a Fibre Channel Protocol (FCP) host

The following is an example of how to configure an FCP host adapter with N_Port Identifier Virtualization (NPIV) by adding a udev rule.

Procedure

1. Take the following sample udev rule **441-zfcp-host-0.0.8000.rules**:

   ```
   ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.8000", DRIVER=="zfcp",
   GOTO="cfg_zfcp_host_0.0.8000"
   ACTION=="add", SUBSYSTEM=="drivers", KERNEL=="zfcp", TEST=="[ccw/0.0.8000]",
   GOTO="cfg_zfcp_host_0.0.8000"
   GOTO="end_zfcp_host_0.0.8000"

   LABEL="cfg_zfcp_host_0.0.8000"
   ATTR{[ccw/0.0.8000]online}="1"

   LABEL="end_zfcp_host_0.0.8000"
   ```

2. Convert the rule to Base64 encoded by running the following command:

   ```
   $ base64 /path/to/file/
   ```

3. Copy the following MCO sample profile into a YAML file:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: worker0  1
     name: 99-worker0-devices
   spec:
     config:
       ignition:
         version: 3.2.0
       storage:
         files:
         - contents:
             source: data:text/plain;base64,<encoded_base64_string>  2
           filesystem: root
           mode: 420
           path: /etc/udev/rules.d/41-zfcp-host-0.0.8000.rules  3
   ```

   1   The role you have defined in the machine config file.

2   The Base64 encoded string that you have generated in the previous step.

3   The path where the udev rule is located.

14.1.2. Configuring an FCP LUN

The following is an example of how to configure an FCP LUN by adding a udev rule. You can add new FCP LUNs or add additional paths to LUNs that are already configured with multipathing.

Procedure

1. Take the following sample udev rule **41-zfcp-lun-0.0.8000:0x500507680d760026:0x00bc000000000000.rules**:

   ```
   ACTION=="add", SUBSYSTEMS=="ccw", KERNELS=="0.0.8000",
   GOTO="start_zfcp_lun_0.0.8207"
   GOTO="end_zfcp_lun_0.0.8000"

   LABEL="start_zfcp_lun_0.0.8000"
   SUBSYSTEM=="fc_remote_ports", ATTR{port_name}=="0x500507680d760026",
   GOTO="cfg_fc_0.0.8000_0x500507680d760026"
   GOTO="end_zfcp_lun_0.0.8000"

   LABEL="cfg_fc_0.0.8000_0x500507680d760026"
   ATTR{[ccw/0.0.8000]0x500507680d760026/unit_add}="0x00bc000000000000"
   GOTO="end_zfcp_lun_0.0.8000"

   LABEL="end_zfcp_lun_0.0.8000"
   ```

2. Convert the rule to Base64 encoded by running the following command:

   ```
   $ base64 /path/to/file/
   ```

3. Copy the following MCO sample profile into a YAML file:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: worker0    1
     name: 99-worker0-devices
   spec:
     config:
       ignition:
         version: 3.2.0
       storage:
         files:
         - contents:
             source: data:text/plain;base64,<encoded_base64_string>    2
           filesystem: root
           mode: 420
           path: /etc/udev/rules.d/41-zfcp-lun-
   0.0.8000:0x500507680d760026:0x00bc000000000000.rules    3
   ```

1 The role you have defined in the machine config file.

2 The Base64 encoded string that you have generated in the previous step.

3 The path where the udev rule is located.

14.1.3. Configuring DASD

The following is an example of how to configure a DASD device by adding a udev rule.

Procedure

1. Take the following sample udev rule **41-dasd-eckd-0.0.4444.rules**:

   ```
   ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.4444", DRIVER=="dasd-eckd",
   GOTO="cfg_dasd_eckd_0.0.4444"
   ACTION=="add", SUBSYSTEM=="drivers", KERNEL=="dasd-eckd", TEST=="
   [ccw/0.0.4444]", GOTO="cfg_dasd_eckd_0.0.4444"
   GOTO="end_dasd_eckd_0.0.4444"

   LABEL="cfg_dasd_eckd_0.0.4444"
   ATTR{[ccw/0.0.4444]online}="1"

   LABEL="end_dasd_eckd_0.0.4444"
   ```

2. Convert the rule to Base64 encoded by running the following command:

   ```
   $ base64 /path/to/file/
   ```

3. Copy the following MCO sample profile into a YAML file:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: worker0   1
     name: 99-worker0-devices
   spec:
     config:
       ignition:
         version: 3.2.0
       storage:
         files:
         - contents:
             source: data:text/plain;base64,<encoded_base64_string>   2
           filesystem: root
           mode: 420
           path: /etc/udev/rules.d/41-dasd-eckd-0.0.4444.rules   3
   ```

1 The role you have defined in the machine config file.

2 The Base64 encoded string that you have generated in the previous step.

3 The path where the udev rule is located.

14.1.4. Configuring qeth

The following is an example of how to configure a qeth device by adding a udev rule.

Procedure

1. Take the following sample udev rule **41-qeth-0.0.1000.rules**:

   ```
   ACTION=="add", SUBSYSTEM=="drivers", KERNEL=="qeth",
   GOTO="group_qeth_0.0.1000"
   ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.1000", DRIVER=="qeth",
   GOTO="group_qeth_0.0.1000"
   ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.1001", DRIVER=="qeth",
   GOTO="group_qeth_0.0.1000"
   ACTION=="add", SUBSYSTEM=="ccw", KERNEL=="0.0.1002", DRIVER=="qeth",
   GOTO="group_qeth_0.0.1000"
   ACTION=="add", SUBSYSTEM=="ccwgroup", KERNEL=="0.0.1000", DRIVER=="qeth",
   GOTO="cfg_qeth_0.0.1000"
   GOTO="end_qeth_0.0.1000"

   LABEL="group_qeth_0.0.1000"
   TEST=="[ccwgroup/0.0.1000]", GOTO="end_qeth_0.0.1000"
   TEST!="[ccw/0.0.1000]", GOTO="end_qeth_0.0.1000"
   TEST!="[ccw/0.0.1001]", GOTO="end_qeth_0.0.1000"
   TEST!="[ccw/0.0.1002]", GOTO="end_qeth_0.0.1000"
   ATTR{[drivers/ccwgroup:qeth]group}="0.0.1000,0.0.1001,0.0.1002"
   GOTO="end_qeth_0.0.1000"

   LABEL="cfg_qeth_0.0.1000"
   ATTR{[ccwgroup/0.0.1000]online}="1"

   LABEL="end_qeth_0.0.1000"
   ```

2. Convert the rule to Base64 encoded by running the following command:

   ```
   $ base64 /path/to/file/
   ```

3. Copy the following MCO sample profile into a YAML file:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfig
   metadata:
     labels:
       machineconfiguration.openshift.io/role: worker0    1
     name: 99-worker0-devices
   spec:
     config:
       ignition:
         version: 3.2.0
       storage:
         files:
   ```

```
        - contents:
            source: data:text/plain;base64,<encoded_base64_string> 2
          filesystem: root
          mode: 420
          path: /etc/udev/rules.d/41-dasd-eckd-0.0.4444.rules 3
```

1   The role you have defined in the machine config file.

2   The Base64 encoded string that you have generated in the previous step.

3   The path where the udev rule is located.

Next steps

- Install and configure the Local Storage Operator

- Updating node network configuration

14.2. CONFIGURING ADDITIONAL DEVICES MANUALLY

Tasks in this section describe how to manually configure additional devices in an IBM Z or IBM® LinuxONE environment. This configuration method is persistent over node restarts but not OpenShift Container Platform native and you need to redo the steps if you replace the node.

Prerequisites

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

- The device must be available to the node.

- In a z/VM environment, the device must be attached to the z/VM guest.

Procedure

1. Connect to the node via SSH by running the following command:

   ```
   $ ssh <user>@<node_ip_address>
   ```

   You can also start a debug session to the node by running the following command:

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

2. To enable the devices with the **chzdev** command, enter the following command:

   ```
   $ sudo chzdev -e 0.0.8000
     sudo chzdev -e 1000-1002
     sude chzdev -e 4444
     sudo chzdev -e 0.0.8000:0x500507680d760026:0x00bc000000000000
   ```

Additional resources

See Persistent device configuration in IBM Documentation.

14.3. ROCE NETWORK CARDS

RoCE (RDMA over Converged Ethernet) network cards do not need to be enabled and their interfaces can be configured with the Kubernetes NMState Operator whenever they are available in the node. For example, RoCE network cards are available if they are attached in a z/VM environment or passed through in a RHEL KVM environment.

14.4. ENABLING MULTIPATHING FOR FCP LUNS

Tasks in this section describe how to manually configure additional devices in an IBM Z or IBM® LinuxONE environment. This configuration method is persistent over node restarts but not OpenShift Container Platform native and you need to redo the steps if you replace the node.

> **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*.

Prerequisites

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

- You have configured multiple paths to a LUN with either method explained above.

Procedure

1. Connect to the node via SSH by running the following command:

   ```
   $ ssh <user>@<node_ip_address>
   ```

   You can also start a debug session to the node by running the following command:

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

2. To enable multipathing, run the following command:

   ```
   $ sudo /sbin/mpathconf --enable
   ```

3. To start the **multipathd** daemon, run the following command:

   ```
   $ sudo multipath
   ```

4. Optional: To format your multipath device with fdisk, run the following command:

   ```
   $ sudo fdisk /dev/mapper/mpatha
   ```

Verification

- To verify that the devices have been grouped, run the following command:

```
$ sudo multipath -ll
```

Example output

```
mpatha (20017380030290197) dm-1 IBM,2810XIV
   size=512G features='1 queue_if_no_path' hwhandler='1 alua' wp=rw
 -+- policy='service-time 0' prio=50 status=enabled
  |- 1:0:0:6  sde 68:16  active ready running
  |- 1:0:1:6  sdf 69:24  active ready running
  |- 0:0:0:6  sdg  8:80  active ready running
  `- 0:0:1:6  sdh 66:48  active ready running
```

Next steps

- Install and configure the Local Storage Operator

- Updating node network configuration

CHAPTER 15. 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.

You create a custom layered image by using a Containerfile and applying it to nodes by using a **MachineConfig** object. 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. You can remove the custom layered image by deleting the machine config, The MCO reboots the nodes back to the base RHCOS image.

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.

> **IMPORTANT**
>
> 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 .

Currently, RHCOS image layering allows you to 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.

In the event you want a Hotfix, it will be 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.

> **NOTE**
>
> It is planned for future releases that you can use RHCOS image layering to incorporate third-party software packages such as libreswan or numactl.

To apply a custom layered image, you create a Containerfile that references an OpenShift Container Platform image and the Hotfix that you want to apply. For example:

Example 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
```

> **NOTE**
>
> Use the same base RHCOS image installed on the rest of your cluster. Use the **oc adm release info --image-for rhel-coreos-8** command to obtain the base image used in your cluster.

Push the resulting custom layered image to an image registry. In a non-production OpenShift Container Platform cluster, create a **MachineConfig** object for the targeted node pool that points to the new image.

The Machine Config Operator (MCO) updates the operating system with content provided in the machine config. This creates a custom layered image that overrides the base RHCOS image on those nodes.

After you create the machine config, the MCO:

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.

15.1. APPLYING A RHCOS 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-8** 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 4.12 image and overrides the kernel package with one from CentOS 8 Stream:

Example Containerfile for a custom layer image

```
# Using a 4.12.0 image
FROM quay.io/openshift-release/ocp-release@sha256... 1
#Install hotfix rpm
RUN rpm-ostree cliwrap install-to-root / \ 2
    rpm-ostree override replace http://mirror.centos.org/centos/8-
stream/BaseOS/x86_64/os/Packages/kernel-{,core-,modules-,modules-extra-}4.18.0-
483.el8.x86_64.rpm && \ 3
    rpm-ostree cleanup -m && \
    ostree container commit
```

1 Specifies the RHCOS base image of your cluster.

2 Enables **cliwrap**. This is currently required to intercept some command invocations made from kernel scripts.

3 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 1
        name: os-layer-hotfix
      spec:
        osImageURL: quay.io/my-registry/custom-image@sha256... 2
      ```

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

      2   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-hotfix                                                           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
```

 ❶

❶ New machine config

❷ 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: 4.12.0-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
```

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 rolled out to the new machine config.

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
v1.25.0+3ef6ef3
ip-10-0-155-125.us-west-1.compute.internal  Ready,SchedulingDisabled  worker
35m  v1.25.0+3ef6ef3
ip-10-0-170-47.us-west-1.compute.internal   Ready            control-plane,master
42m  v1.25.0+3ef6ef3
ip-10-0-174-77.us-west-1.compute.internal   Ready            control-plane,master
42m  v1.25.0+3ef6ef3
ip-10-0-211-49.us-west-1.compute.internal   Ready            control-plane,master
42m  v1.25.0+3ef6ef3
ip-10-0-218-151.us-west-1.compute.internal  Ready             worker          31m
v1.25.0+3ef6ef3
```

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](#)

15.2. 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-hotfix
   ```

   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  1
   ```

   1 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
v1.25.0+3ef6ef3
ip-10-0-155-125.us-west-1.compute.internal  Ready,SchedulingDisabled  worker
35m   v1.25.0+3ef6ef3
ip-10-0-170-47.us-west-1.compute.internal   Ready                control-plane,master  42m
v1.25.0+3ef6ef3
ip-10-0-174-77.us-west-1.compute.internal   Ready                control-plane,master  42m
v1.25.0+3ef6ef3
ip-10-0-211-49.us-west-1.compute.internal   Ready                control-plane,master  42m
v1.25.0+3ef6ef3
ip-10-0-218-151.us-west-1.compute.internal  Ready                worker              31m
v1.25.0+3ef6ef3
```

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
   ```

15.3. 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.