



OpenShift Container Platform 4.9

Post-installation configuration

Day 2 operations for OpenShift Container Platform

OpenShift Container Platform 4.9 Post-installation configuration

Day 2 operations for OpenShift Container Platform

Legal Notice

Copyright © 2022 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

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

Table of Contents

CHAPTER 1. POST-INSTALLATION CONFIGURATION OVERVIEW	8
1.1. PERFORMING POST-INSTALLATION CONFIGURATION TASKS	8
CHAPTER 2. CONFIGURING A PRIVATE CLUSTER	10
2.1. ABOUT PRIVATE CLUSTERS	10
DNS	10
Ingress Controller	10
API server	10
2.2. SETTING DNS TO PRIVATE	10
2.3. SETTING THE INGRESS CONTROLLER TO PRIVATE	12
2.4. RESTRICTING THE API SERVER TO PRIVATE	12
CHAPTER 3. POST-INSTALLATION MACHINE CONFIGURATION TASKS	15
3.1. UNDERSTANDING THE MACHINE CONFIG OPERATOR	15
3.1.1. Machine Config Operator	15
Purpose	15
Project	15
3.1.2. Machine config overview	15
3.1.2.1. What can you change with machine configs?	16
3.1.2.2. Project	17
3.1.3. Checking machine config pool status	17
3.2. USING MACHINECONFIG OBJECTS TO CONFIGURE NODES	19
3.2.1. Configuring chrony time service	19
3.2.2. Disabling the chrony time service	21
3.2.3. Adding kernel arguments to nodes	22
3.2.4. Enabling multipathing with kernel arguments on RHCOS	25
3.2.5. Adding a real-time kernel to nodes	27
3.2.6. Configuring journald settings	29
3.2.7. Adding extensions to RHCOS	30
3.2.8. Loading custom firmware blobs in the machine config manifest	32
3.3. CONFIGURING MCO-RELATED CUSTOM RESOURCES	33
3.3.1. Creating a KubeletConfig CRD to edit kubelet parameters	33
3.3.2. Creating a ContainerRuntimeConfig CR to edit CRI-O parameters	38
3.3.3. Setting the default maximum container root partition size for Overlay with CRI-O	42
CHAPTER 4. POST-INSTALLATION CLUSTER TASKS	45
4.1. AVAILABLE CLUSTER CUSTOMIZATIONS	45
4.1.1. Cluster configuration resources	45
4.1.2. Operator configuration resources	46
4.1.3. Additional configuration resources	46
4.1.4. Informational Resources	47
4.2. UPDATING THE GLOBAL CLUSTER PULL SECRET	47
4.3. ADJUST WORKER NODES	49
4.3.1. Understanding the difference between machine sets and the machine config pool	49
4.3.2. Scaling a machine set manually	49
4.3.3. The machine set deletion policy	50
4.3.4. Creating default cluster-wide node selectors	50
4.4. CREATING INFRASTRUCTURE MACHINE SETS FOR PRODUCTION ENVIRONMENTS	54
4.4.1. Creating a machine set	54
4.4.2. Creating an infrastructure node	56
4.4.3. Creating a machine config pool for infrastructure machines	57
4.5. ASSIGNING MACHINE SET RESOURCES TO INFRASTRUCTURE NODES	60

4.5.1. Binding infrastructure node workloads using taints and tolerations	60
4.6. MOVING RESOURCES TO INFRASTRUCTURE MACHINE SETS	62
4.6.1. Moving the router	62
4.6.2. Moving the default registry	64
4.6.3. Moving the monitoring solution	65
4.6.4. Moving OpenShift Logging resources	66
4.7. ABOUT THE CLUSTER AUTOSCALER	70
4.7.1. ClusterAutoscaler resource definition	72
4.7.2. Deploying the cluster autoscaler	73
4.8. ABOUT THE MACHINE AUTOSCALER	74
4.8.1. MachineAutoscaler resource definition	74
4.8.2. Deploying the machine autoscaler	75
4.9. ENABLING TECHNOLOGY PREVIEW FEATURES USING FEATUREGATES	75
4.9.1. Understanding feature gates	75
4.9.2. Enabling feature sets using the CLI	76
4.10. ETCD TASKS	77
4.10.1. About etcd encryption	77
4.10.2. Enabling etcd encryption	77
4.10.3. Disabling etcd encryption	79
4.10.4. Backing up etcd data	80
4.10.5. Defragmenting etcd data	82
4.10.5.1. Automatic defragmentation	82
4.10.5.2. Manual defragmentation	82
4.10.6. Restoring to a previous cluster state	85
4.10.7. Issues and workarounds for restoring a persistent storage state	95
4.11. POD DISRUPTION BUDGETS	96
4.11.1. Understanding how to use pod disruption budgets to specify the number of pods that must be up	96
4.11.2. Specifying the number of pods that must be up with pod disruption budgets	97
4.12. ROTATING OR REMOVING CLOUD PROVIDER CREDENTIALS	98
4.12.1. Rotating cloud provider credentials manually	98
4.12.2. Removing cloud provider credentials	100
4.13. CONFIGURING IMAGE STREAMS FOR A DISCONNECTED CLUSTER	101
4.13.1. Cluster Samples Operator assistance for mirroring	101
4.13.2. Using Cluster Samples Operator image streams with alternate or mirrored registries	101
4.13.3. Preparing your cluster to gather support data	103
Additional resources	103
CHAPTER 5. POST-INSTALLATION NODE TASKS	104
5.1. ADDING RHEL COMPUTE MACHINES TO AN OPENSIFT CONTAINER PLATFORM CLUSTER	104
5.1.1. About adding RHEL compute nodes to a cluster	104
5.1.2. System requirements for RHEL compute nodes	104
5.1.2.1. Certificate signing requests management	105
5.1.3. Preparing the machine to run the playbook	106
5.1.4. Preparing a RHEL compute node	107
5.1.5. Adding a RHEL compute machine to your cluster	108
5.1.6. Required parameters for the Ansible hosts file	109
5.1.7. Optional: Removing RHCOS compute machines from a cluster	110
5.2. ADDING RHCOS COMPUTE MACHINES TO AN OPENSIFT CONTAINER PLATFORM CLUSTER	111
5.2.1. Prerequisites	111
5.2.2. Creating more RHCOS machines using an ISO image	111
5.2.3. Creating more RHCOS machines by PXE or iPXE booting	112
5.2.4. Approving the certificate signing requests for your machines	113
5.3. DEPLOYING MACHINE HEALTH CHECKS	116

5.3.1. About machine health checks	116
5.3.1.1. Limitations when deploying machine health checks	117
5.3.2. Sample MachineHealthCheck resource	117
5.3.2.1. Short-circuiting machine health check remediation	118
5.3.2.1.1. Setting maxUnhealthy by using an absolute value	119
5.3.2.1.2. Setting maxUnhealthy by using percentages	119
5.3.3. Creating a MachineHealthCheck resource	119
5.3.4. Scaling a machine set manually	119
5.3.5. Understanding the difference between machine sets and the machine config pool	120
5.4. RECOMMENDED NODE HOST PRACTICES	121
5.4.1. Creating a KubeletConfig CRD to edit kubelet parameters	121
5.4.2. Modifying the number of unavailable worker nodes	126
5.4.3. Control plane node sizing	126
5.4.4. Setting up CPU Manager	128
5.5. HUGE PAGES	132
5.5.1. What huge pages do	132
5.5.2. How huge pages are consumed by apps	133
5.5.3. Configuring huge pages	134
5.5.3.1. At boot time	134
5.6. UNDERSTANDING DEVICE PLUG-INS	135
Example device plug-ins	136
5.6.1. Methods for deploying a device plug-in	136
5.6.2. Understanding the Device Manager	137
5.6.3. Enabling Device Manager	137
5.7. TAINTS AND TOLERATIONS	139
5.7.1. Understanding taints and tolerations	139
5.7.1.1. Understanding how to use toleration seconds to delay pod evictions	141
5.7.1.2. Understanding how to use multiple taints	142
5.7.1.3. Understanding pod scheduling and node conditions (taint node by condition)	143
5.7.1.4. Understanding evicting pods by condition (taint-based evictions)	143
5.7.1.5. Tolerating all taints	145
5.7.2. Adding taints and tolerations	145
5.7.3. Adding taints and tolerations using a machine set	146
5.7.4. Binding a user to a node using taints and tolerations	148
5.7.5. Controlling nodes with special hardware using taints and tolerations	149
5.7.6. Removing taints and tolerations	150
5.8. TOPOLOGY MANAGER	151
5.8.1. Topology Manager policies	151
5.8.2. Setting up Topology Manager	151
5.8.3. Pod interactions with Topology Manager policies	152
5.9. RESOURCE REQUESTS AND OVERCOMMITMENT	153
5.10. CLUSTER-LEVEL OVERCOMMIT USING THE CLUSTER RESOURCE OVERRIDE OPERATOR	153
5.10.1. Installing the Cluster Resource Override Operator using the web console	154
5.10.2. Installing the Cluster Resource Override Operator using the CLI	156
5.10.3. Configuring cluster-level overcommit	159
5.11. NODE-LEVEL OVERCOMMIT	160
5.11.1. Understanding compute resources and containers	161
5.11.1.1. Understanding container CPU requests	161
5.11.1.2. Understanding container memory requests	161
5.11.2. Understanding overcommitment and quality of service classes	161
5.11.2.1. Understanding how to reserve memory across quality of service tiers	162
5.11.3. Understanding swap memory and QOS	162
5.11.4. Understanding nodes overcommitment	163

5.11.5. Disabling or enforcing CPU limits using CPU CFS quotas	163
5.11.6. Reserving resources for system processes	165
5.11.7. Disabling overcommitment for a node	165
5.12. PROJECT-LEVEL LIMITS	166
5.12.1. Disabling overcommitment for a project	166
5.13. FREEING NODE RESOURCES USING GARBAGE COLLECTION	166
5.13.1. Understanding how terminated containers are removed through garbage collection	166
5.13.2. Understanding how images are removed through garbage collection	167
5.13.3. Configuring garbage collection for containers and images	168
5.14. USING THE NODE TUNING OPERATOR	170
5.14.1. Accessing an example Node Tuning Operator specification	171
5.14.2. Custom tuning specification	171
5.14.3. Default profiles set on a cluster	175
5.14.4. Supported TuneD daemon plug-ins	176
5.15. CONFIGURING THE MAXIMUM NUMBER OF PODS PER NODE	177
CHAPTER 6. POST-INSTALLATION NETWORK CONFIGURATION	180
6.1. CLUSTER NETWORK OPERATOR CONFIGURATION	180
6.2. ENABLING THE CLUSTER-WIDE PROXY	180
6.3. SETTING DNS TO PRIVATE	182
6.4. CONFIGURING INGRESS CLUSTER TRAFFIC	183
6.5. CONFIGURING THE NODE PORT SERVICE RANGE	184
6.5.1. Prerequisites	184
6.5.1.1. Expanding the node port range	184
6.6. CONFIGURING NETWORK POLICY	185
6.6.1. About network policy	185
6.6.2. Example NetworkPolicy object	188
6.6.3. Creating a network policy	188
6.6.4. Configuring multitenant isolation by using network policy	190
6.6.5. Creating default network policies for a new project	192
6.6.6. Modifying the template for new projects	192
6.6.6.1. Adding network policies to the new project template	193
6.7. SUPPORTED CONFIGURATIONS	194
6.7.1. Supported network configurations	195
6.7.2. Supported configurations for Kiali	195
6.7.3. Supported configurations for Distributed Tracing	195
6.7.4. Supported Mixer adapters	195
6.7.5. Operator overview	195
6.8. OPTIMIZING ROUTING	196
6.8.1. Baseline Ingress Controller (router) performance	196
6.8.2. Ingress Controller (router) performance optimizations	197
6.9. POST-INSTALLATION RHOSP NETWORK CONFIGURATION	198
6.9.1. Configuring application access with floating IP addresses	198
6.9.2. Kuryr ports pools	199
6.9.3. Adjusting Kuryr ports pool settings in active deployments on RHOSP	199
CHAPTER 7. POST-INSTALLATION STORAGE CONFIGURATION	201
7.1. DYNAMIC PROVISIONING	201
7.1.1. About dynamic provisioning	201
7.1.2. Available dynamic provisioning plug-ins	201
7.2. DEFINING A STORAGE CLASS	202
7.2.1. Basic StorageClass object definition	203
7.2.2. Storage class annotations	203

7.2.3. RHOSP Cinder object definition	204
7.2.4. AWS Elastic Block Store (EBS) object definition	204
7.2.5. Azure Disk object definition	205
7.2.6. Azure File object definition	206
7.2.6.1. Considerations when using Azure File	207
7.2.7. GCE PersistentDisk (gcePD) object definition	208
7.2.8. VMware vSphere object definition	208
7.2.9. Red Hat Virtualization (RHV) object definition	209
7.3. CHANGING THE DEFAULT STORAGE CLASS	210
7.4. OPTIMIZING STORAGE	210
7.5. AVAILABLE PERSISTENT STORAGE OPTIONS	211
7.6. RECOMMENDED CONFIGURABLE STORAGE TECHNOLOGY	212
7.6.1. Specific application storage recommendations	212
7.6.1.1. Registry	213
7.6.1.2. Scaled registry	213
7.6.1.3. Metrics	213
7.6.1.4. Logging	214
7.6.1.5. Applications	214
7.6.2. Other specific application storage recommendations	214
7.7. DEPLOY RED HAT OPENSIFT CONTAINER STORAGE	214
CHAPTER 8. PREPARING FOR USERS	216
8.1. UNDERSTANDING IDENTITY PROVIDER CONFIGURATION	216
8.1.1. About identity providers in OpenShift Container Platform	216
8.1.2. Supported identity providers	216
8.1.3. Identity provider parameters	217
8.1.4. Sample identity provider CR	217
8.2. USING RBAC TO DEFINE AND APPLY PERMISSIONS	218
8.2.1. RBAC overview	218
8.2.1.1. Default cluster roles	219
8.2.1.2. Evaluating authorization	221
8.2.1.2.1. Cluster role aggregation	221
8.2.2. Projects and namespaces	222
8.2.3. Default projects	222
8.2.4. Viewing cluster roles and bindings	223
8.2.5. Viewing local roles and bindings	229
8.2.6. Adding roles to users	231
8.2.7. Creating a local role	233
8.2.8. Creating a cluster role	234
8.2.9. Local role binding commands	234
8.2.10. Cluster role binding commands	235
8.2.11. Creating a cluster admin	235
8.3. THE KUBEADMIN USER	236
8.3.1. Removing the kubeadmin user	236
8.4. IMAGE CONFIGURATION	237
8.4.1. Image controller configuration parameters	237
8.4.2. Configuring image registry settings	239
8.4.2.1. Configuring additional trust stores for image registry access	241
8.4.2.2. Configuring image registry repository mirroring	242
8.5. POPULATING OPERATORHUB FROM MIRRORED OPERATOR CATALOGS	245
8.5.1. Prerequisites	245
8.5.2. Creating the ImageContentSourcePolicy object	245
8.5.3. Adding a catalog source to a cluster	246

8.6. ABOUT OPERATOR INSTALLATION WITH OPERATORHUB	247
8.6.1. Installing from OperatorHub using the web console	248
8.6.2. Installing from OperatorHub using the CLI	249
CHAPTER 9. CONFIGURING ALERT NOTIFICATIONS	252
9.1. SENDING NOTIFICATIONS TO EXTERNAL SYSTEMS	252
9.1.1. Configuring alert receivers	252
9.2. ADDITIONAL RESOURCES	253
CHAPTER 10. CONVERTING A CONNECTED CLUSTER TO A DISCONNECTED CLUSTER	255
10.1. ABOUT THE MIRROR REGISTRY	255
10.2. PREREQUISITES	256
10.3. PREPARING THE CLUSTER FOR MIRRORING	256
10.4. MIRRORING THE IMAGES	258
10.5. CONFIGURING THE CLUSTER FOR THE MIRROR REGISTRY	260
10.6. ENSURE APPLICATIONS CONTINUE TO WORK	263
10.7. DISCONNECT THE CLUSTER FROM THE NETWORK	264
10.8. RESTORING A DEGRADED INSIGHTS OPERATOR	264
10.9. RESTORING THE NETWORK	264

CHAPTER 1. POST-INSTALLATION CONFIGURATION OVERVIEW

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

- Machine
- Cluster
- Node
- Network
- Storage
- Users
- Alerts and notifications

1.1. PERFORMING POST-INSTALLATION CONFIGURATION TASKS

Cluster administrators can perform the following post-installation configuration tasks:

- [Configure operating system features](#): Machine Config Operator (MCO) manages **MachineConfig** objects. By using MCO, you can perform the following on an OpenShift Container Platform cluster:
 - Configure nodes by using **MachineConfig** objects
 - Configure MCO-related custom resources
- [Configure cluster features](#): As a cluster administrator, you can modify the configuration resources of the major features of an OpenShift Container Platform cluster. These features include:
 - Image registry
 - Networking configuration
 - Image build behavior
 - Identity provider
 - The etcd configuration
 - Machine set creation to handle the workloads
 - Cloud provider credential management
- [Configure cluster components to be private](#): By default, the installation program provisions OpenShift Container Platform by using a publicly accessible DNS and endpoints. If you want your cluster to be accessible from within an internal network only, configure the following components to be private:
 - DNS

- Ingress Controller
- API server
- [Perform node operations](#): By default, OpenShift Container Platform uses Red Hat Enterprise Linux CoreOS (RHCOS) compute machines. As a cluster administrator, you can perform the following operations with the machines in your OpenShift Container Platform cluster:
 - Add and remove compute machines
 - Add and remove taints and tolerations to the nodes
 - Configure the maximum number of pods per node
 - Enable Device Manager
- [Configure network](#): After installing OpenShift Container Platform, as a cluster administrator, you can configure the following:
 - Ingress cluster traffic
 - Node port service range
 - Network policy
 - Enabling the cluster-wide proxy
- [Configure storage](#): By default, containers operate using ephemeral storage or transient local storage. The ephemeral storage has a lifetime limitation, so you must configure persistent storage to store the data for a long time. 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**: Cluster administrators can use Kubernetes persistent volumes to make existing storage available to a cluster by supporting various device configurations and mount options.
- [Configure users](#): OAuth access tokens allow users to authenticate themselves to the API. As a cluster administrator, you can configure OAuth to specify an identity provider, use role-based access control to define and apply permissions to users, and install an Operator from OperatorHub.
- [Manage alerts and notifications](#): As a cluster administrator, you can view firing alerts by default from the Alerting UI of the web console. You can also configure OpenShift Container Platform to send alert notifications to external systems so that you learn about important issues with your cluster.

CHAPTER 2. CONFIGURING A PRIVATE CLUSTER

After you install an OpenShift Container Platform version 4.9 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 cluster.

DNS

If you install OpenShift Container Platform on installer-provisioned infrastructure, the installation program creates records in a pre-existing public zone and, where possible, creates a private zone for the cluster's own DNS resolution. In both the public zone and the private zone, the installation program or cluster creates DNS entries for ***.apps**, for 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. You can replace the default Ingress Controller with an internal one.

API server

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

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

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

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

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.

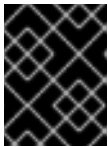
2. Patch the **DNS** custom resource to remove the public zone:

```

$ 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>-wfp4: 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.

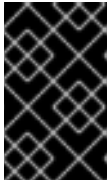
Prerequisites

- 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 AWS or Azure, take the following actions:
 - a. Locate and delete appropriate load balancer component.
 - For AWS, delete the external load balancer. The API DNS entry in the private zone already points to the internal load balancer, which uses an identical configuration, so you do not need to modify the internal load balancer.
 - For Azure, delete the **api-internal** rule for the load balancer.

- b. Delete the **api.\$clustername.\$yourdomain** DNS entry in the public zone.
2. 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.

- a. From your terminal, list the cluster machines:

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

Example output

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

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

- b. Remove the external load balancer from each control plane machine.
 - i. Edit a control plane **Machine** object to remove the reference to the external load balancer:

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

- ❶ Specify the name of the control plane, or master, **Machine** object to modify.

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

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

- ❶ ❷ Delete this line.

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

CHAPTER 3. POST-INSTALLATION 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.

3.1. UNDERSTANDING THE MACHINE CONFIG OPERATOR

3.1.1. 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.

Project

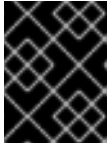
[openshift-machine-config-operator](#)

3.1.2. 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:

- 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.

- 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 post-installation 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.

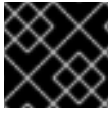
MCO uses [Ignition](#) as the configuration format. OpenShift Container Platform 4.6 moved from Ignition config specification version 2 to version 3.

3.1.2.1. What can you change with machine configs?

The kinds of components that MCO can change include:

- **config**: Create Ignition config objects (see the [Ignition configuration specification](#)) to do things like modify files, systemd services, and other features on OpenShift Container Platform machines, including:
 - **Configuration files**: Create or overwrite files in the **/var** or **/etc** directory.

- **systemd units:** Create and set the status of a systemd service or add to an existing systemd service by dropping in additional settings.
- **users and groups:** Change SSH keys in the passwd section post-installation.



IMPORTANT

Changing SSH keys via machine configs is only supported for the **core** user.

- **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 post-installation procedure.



IMPORTANT

The use of FIPS Validated / Modules in Process cryptographic libraries is only supported on OpenShift Container Platform deployments on the **x86_64** architecture.

- **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 post-installation are included in the following procedures. See descriptions of RHCOS bare metal installation for system configuration tasks that must be done during or before OpenShift Container Platform installation.

3.1.2.2. Project

See the [openshift-machine-config-operator](#) GitHub site for details.

3.1.3. Checking machine config pool status

To see the status of the Machine Config Operator, 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 pool, type:

```
$ oc get machineconfigpool
NAME      CONFIG      UPDATED  UPDATING  DEGRADED  MACHINECOUNT
```

```

READYMACHINECOUNT UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
AGE
master rendered-master-dd... True False False 3 3 3
0 4h42m
worker rendered-worker-fde... True False False 3 3 3
0 4h42m

```

In the previous output, there are three master and three worker nodes. All machines are updated and none are currently updating. Because all nodes are **Updated** and **Ready** and none are **Degraded**, you can tell that there are no issues.

- To see each existing **machineconfig**, type:

```

$ oc get machineconfigs
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 **machineconfigs** listed as **rendered** are not meant to be changed or deleted. Expect them to be hidden at some point in the future.

- Check the status of worker (or change to master) to see the status of that pool of nodes:

```

$ oc describe mcp worker
...
Degraded Machine Count: 0
Machine Count: 3
Observed Generation: 2
Ready Machine Count: 3
Unavailable Machine Count: 0
Updated Machine Count: 3
Events: <none>

```

- You can view the contents of a particular machine config (in this case, **01-master-kubelet**). The trimmed output from the following **oc describe** command shows that this **machineconfig** contains both configuration files (**cloud.conf** and **kubelet.conf**) and a systemd service (Kubernetes Kubelet):

```

$ oc describe machineconfigs 01-master-kubelet
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%2Fv1beta1%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 typing:

```
$ 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.

3.2. 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 [adding](#) or [updating](#) SSH authorized keys, [verifying image signatures](#), [enabling SCTP](#), and [configuring iSCSI initiator names](#) 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.

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.

3.2.1. 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: 4.9.0
metadata:
  name: 99-worker-chrony ❶
  labels:
    machineconfiguration.openshift.io/role: worker ❷
storage:
  files:
  - path: /etc/chrony.conf
    mode: 0644
    overwrite: true
  contents:
    inline: |
      pool 0.rhel.pool.ntp.org iburst ❸
      driftfile /var/lib/chrony/drift
      makestep 1.0 3
      rtcsync
      logdir /var/log/chrony
```

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


3.2.2. 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=ntpd.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
```

■

3.2.3. 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:

- **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 recommended for production systems, permissive mode can be helpful for debugging.
- **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.

See [Kernel.org kernel parameters](https://kernel.org/doc/Documentation/kernel-parameters.txt) 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:
  config:
    ignition:
      version: 3.2.0
  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	GENERATEDBY	CONTROLLER
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

- Check the nodes:

```
$ oc get nodes
```

Example output

NAME	STATUS	ROLES	AGE	VERSION
ip-10-0-136-161.ec2.internal	Ready	worker	28m	v1.22.1
ip-10-0-136-243.ec2.internal	Ready	master	34m	v1.22.1
ip-10-0-141-105.ec2.internal	Ready,SchedulingDisabled	worker	28m	v1.22.1
ip-10-0-142-249.ec2.internal	Ready	master	34m	v1.22.1
ip-10-0-153-11.ec2.internal	Ready	worker	28m	v1.22.1
ip-10-0-153-150.ec2.internal	Ready	master	34m	v1.22.1

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

- 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
```

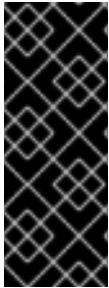
```
rootflags=defaults,priquota rw root=UUID=fd0... ostree=/ostree/boot.0/rhcos/16...
coreos.oem.id=qemu coreos.oem.id=ec2 ignition.platform.id=ec2 enforcing=0
```

```
sh-4.2# exit
```

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

3.2.4. 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. Post-installation 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 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 LinuxONE*.

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.

Procedure

1. To enable multipathing post-installation 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 post-installation 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-master-kargs-mpath.yaml
```

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

```
$ oc get MachineConfig
```

Example output

NAME	GENERATEDBY	CONTROLLER
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-kargs-mpath	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
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	v1.22.1
ip-10-0-136-243.ec2.internal	Ready	master	34m	v1.22.1
ip-10-0-141-105.ec2.internal	Ready,SchedulingDisabled	worker	28m	v1.22.1
ip-10-0-142-249.ec2.internal	Ready	master	34m	v1.22.1
ip-10-0-153-11.ec2.internal	Ready	worker	28m	v1.22.1
ip-10-0-153-150.ec2.internal	Ready	master	34m	v1.22.1

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.

3.2.5. 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, 4.9 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 v1.22.1
ip-10-0-146-92.us-east-2.compute.internal Ready worker 101m v1.22.1
ip-10-0-169-2.us-east-2.compute.internal Ready worker 102m v1.22.1
```

```
$ 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
```

3.2.6. 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: 4.9.0
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
```

3.2.7. 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 extension is available:

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

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	v1.22.1

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

Example output

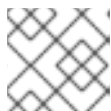
```
...
To use host binaries, run `chroot /host`
sh-4.4# chroot /host
sh-4.4# rpm -q usbguard
usbguard-0.7.4-4.el8.x86_64.rpm
```

3.2.8. 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: 4.9.0
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 98-worker-firmware-blob
storage:
```

```

files:
- path: /var/lib/firmware/<package_name> ❶
  contents:
    local: <package_name> ❷
    mode: 0644 ❸
openshift:
  kernel_arguments:
    - 'firmware_class.path=/var/lib/firmware' ❹

```

- ❶ Sets the path on the node where the firmware package is copied to.
- ❷ 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.
- ❸ Sets the permissions for the file on the RHCOS node. It is recommended to set **0644** permissions.
- ❹ 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.

3.3. 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.

3.3.1. 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:

- Create one **KubeletConfig** 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 of the pools, you need only one **KubeletConfig** CR for all of the pools.
- 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.
- 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**.

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:

```

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

```

- ❶ Enter the label from the machine config pool.
- ❷ 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=large-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
```

3.3.2. 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:** The **pidsLimit** parameter sets the CRI-O **pids_limit** parameter, which is maximum number of processes allowed in a container. The default is 1024 (**pids_limit = 1024**).
- **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. The default is 10 GB (size = "10G").
- **Maximum log size:** The **logSizeMax** parameter sets the CRI-O **log_size_max** parameter, which is the maximum size allowed for the container log file. The default is unlimited (**log_size_max = -1**). If set to a positive number, it must be at least 8192 to not be smaller than the ConMon read buffer. ConMon is a program that monitors communications between a container manager (such as Podman or CRI-O) and the OCI runtime (such as runc or crun) for a single container.

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-pid   24m
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 raises the **pids_limit** to 2048, sets the **log_level** to **debug**, sets the overlay size to 8 GB, and sets the **log_size_max** to unlimited:

Example ContainerRuntimeConfig CR

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

```

pidsLimit: 2048 2
logLevel: debug 3
overlaySize: 8G 4
logSizeMax: "-1" 5

```

- 1 Specifies the machine config pool label.
- 2 Optional: Specifies the maximum number of processes allowed in a container.
- 3 Optional: Specifies the level of verbosity for log messages.
- 4 Optional: Specifies the maximum size of a container image.
- 5 Optional: Specifies the maximum size allowed for the container log file. If set to a positive number, it must be at least 8192.

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
    pidsLimit: 2048
    logLevel: debug
    overlaySize: 8G
    logSizeMax: "-1"

```

- 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	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 | egrep 'log_level|pids_limit|log_size_max'
```

Example output

```
pids_limit = 2048
log_size_max = -1
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"
```

3.3.3. 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 and PID limit, 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:
    pidsLimit: 2048
    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 4173030d89bf4a7a0976d1665491a4d9a6e54f1
3.2.0 7m42s
rendered-worker-xyz 4173030d89bf4a7a0976d1665491a4d9a6e54f1 3.2.0
7m36s
```

- 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 4. POST-INSTALLATION CLUSTER TASKS

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

4.1. 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**

4.1.1. 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 policies and default node selectors.

4.1.2. 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 internal image registry settings such as public routing, log levels, proxy settings, resource constraints, replica counts, and storage type.
config.samples.operator.openshift.io	Configures the Samples Operator to control which example image streams and templates are installed on the cluster.

4.1.3. 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.

4.1.4. 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 4.9, 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 .

4.2. 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.



IMPORTANT

To transfer your cluster to another owner, you must first initiate the transfer in [Red Hat OpenShift Cluster Manager](#), and then update the pull secret on the cluster. Updating a cluster's pull secret without initiating the transfer in OpenShift Cluster Manager causes the cluster to stop reporting Telemetry metrics in OpenShift Cluster Manager.

For more information [about transferring cluster ownership](#), see "Transferring cluster ownership" in the Red Hat OpenShift Cluster Manager documentation.

**WARNING**

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

Prerequisites

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

Procedure

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

- 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.

- 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.

- 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.

4.3. ADJUST WORKER NODES

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

4.3.1. Understanding the difference between 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.

4.3.2. Scaling a machine set manually

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

This guidance is relevant to fully automated, installer-provisioned infrastructure installations. Customized, user-provisioned infrastructure installations do not have 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 machine sets that are in the cluster:

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

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

2. Scale the machine set:

```
$ 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 machine set:

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

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

4.3.3. The 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 machine sets down. The deletion policy can be set according to the use case by modifying the particular 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/cluster-api-delete-machine** 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 machine set to **0** unless you first relocate the router pods.

**NOTE**

Custom 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 machine sets are scaling down. This prevents service disruption.

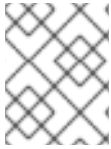
4.3.4. Creating default cluster-wide node selectors

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 machine set, or a machine config. Adding the label to the 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
  policy:
    name: ""
```

- 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 machine set or editing the node directly:
 - Use a machine set to add labels to nodes managed by the 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 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 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. Verify that the labels are added to the **MachineSet** object by using the **oc edit** command:
For example:


```
$ oc edit MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

- e. Redeploy the nodes associated with that 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
```

- f. 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  v1.22.1
```

- 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	v1.22.1

4.4. CREATING INFRASTRUCTURE MACHINE SETS FOR PRODUCTION ENVIRONMENTS

You can create a 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 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 machine sets, one for each availability zone.

For information on infrastructure nodes and which components can run on infrastructure nodes, see [Creating infrastructure machine sets](#).

For sample machine sets that you can use with these procedures, see [Creating machine sets for different clouds](#).

4.4.1. Creating a machine set

In addition to the ones created by the installation program, you can create your own machine sets 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 machine set custom resource (CR) sample and is named **<file_name>.yaml**.
Ensure that you set the **<clusterID>** and **<role>** parameter values.
 - a. If you are not sure which value to set for a specific field, you can check an existing machine set from your cluster:

```
$ 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. Check values of a specific machine set:

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

Example output

```
...
template:
  metadata:
    labels:
      machine.openshift.io/cluster-api-cluster: agl030519-vplxk 1
      machine.openshift.io/cluster-api-machine-role: worker 2
      machine.openshift.io/cluster-api-machine-type: worker
      machine.openshift.io/cluster-api-machineset: agl030519-vplxk-worker-us-east-1a
```

1 The cluster ID.

2 A default node label.

2. Create the new **MachineSet** CR:

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

3. View the list of machine sets:

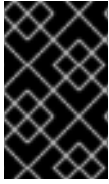
```
$ oc get machineset -n openshift-machine-api
```

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

4.4.2. 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: topology.kubernetes.io/region=us-east-1 1
...
```

- 1** This example node selector deploys pods on nodes in the **us-east-1** region by default.

- c. Save the file to apply the changes.

5. 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](#).

4.4.3. 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	GENERATEDBY	CONTROLLER
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-072d4b2da7f88162636902b074e9e28e5b6fb8349a29735e48446d435962dec4547d3090	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.2.0 31d
rendered-master-3e88ec72aed3886dec061df60d16d1af02c07496ba0417b3e12b78fb32baf6293d314f79	365c1cfd14de5b0e3b85e0fc815b0060f36ab955	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 1
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - path: /etc/infratest
        mode: 0644
      contents:
        source: data:,infra
```

- 1 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	1	0	4m20s	True	False
1						
master	rendered-master-9360fdb895d4c131c7c4bebbae099c90	3	3	0	91m	True
3						
worker	rendered-worker-60e35c2e99f42d976e084fa94da4d0fc	2	2	0	91m	True
2						

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.

4.5. 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.

4.5.1. 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>:<effect>
```

For example:

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

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: NoSchedule
  ...
```

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: NoSchedule 1
  key: node-role.kubernetes.io/infra 2
  operator: Exists 3
```

- 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.

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.

4.6. 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.

4.6.1. Moving the router

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

Prerequisites

- Configure additional 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
```

Add the **nodeSelector** stanza that references the **infra** label to the **spec** section, as shown:

```
spec:
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/infra: ""
```

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  v1.22.1
```

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

4.6.2. Moving the default registry

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

Prerequisites

- Configure additional 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-a524-12f6ee2931b
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
```

3. Modify the **spec** section of the object to resemble the following YAML:

```

spec:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - podAffinityTerm:
            namespaces:
              - openshift-image-registry
            topologyKey: kubernetes.io/hostname
            weight: 100
  logLevel: Normal
  managementState: Managed
  nodeSelector:
    node-role.kubernetes.io/infra: ""

```

4. 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.

4.6.3. Moving the monitoring solution

By default, the Prometheus Cluster Monitoring stack, which contains Prometheus, Grafana, and AlertManager, is deployed to provide cluster monitoring. It is managed by the Cluster Monitoring Operator. To move its components to different machines, you create and apply a custom config map.

Procedure

1. Save the following **ConfigMap** definition as the **cluster-monitoring-configmap.yaml** file:

```
apiVersion: v1
```

```

kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |+
    alertmanagerMain:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    prometheusK8s:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    prometheusOperator:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    grafana:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    k8sPrometheusAdapter:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    kubeStateMetrics:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    telemeterClient:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    openshiftStateMetrics:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    thanosQuerier:
      nodeSelector:
        node-role.kubernetes.io/infra: ""

```

Running this config map forces the components of the monitoring stack to redeploy to infrastructure nodes.

2. Apply the new config map:

```
$ oc create -f cluster-monitoring-configmap.yaml
```

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

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

4. 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.

4.6.4. Moving OpenShift Logging resources

You can configure the Cluster Logging Operator to deploy the pods for OpenShift Logging components, such as Elasticsearch and Kibana, to different nodes. You cannot move the Cluster Logging Operator pod from its installed location.

For example, you can move the Elasticsearch pods to a separate node because of high CPU, memory, and disk requirements.

Prerequisites

- OpenShift Logging and Elasticsearch must be installed. These features are not installed by default.

Procedure

1. Edit the **ClusterLogging** custom resource (CR) in the **openshift-logging** project:

```
$ oc edit ClusterLogging instance
```

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging

...

spec:
  collection:
    logs:
      fluentd:
        resources: null
        type: fluentd
  logStore:
    elasticsearch:
      nodeCount: 3
      nodeSelector: 1
        node-role.kubernetes.io/infra: "
      redundancyPolicy: SingleRedundancy
      resources:
        limits:
          cpu: 500m
          memory: 16Gi
        requests:
          cpu: 500m
          memory: 16Gi
        storage: {}
      type: elasticsearch
    managementState: Managed
  visualization:
    kibana:
      nodeSelector: 2
        node-role.kubernetes.io/infra: "
      proxy:
        resources: null
      replicas: 1
      resources: null
```

```
type: kibana
```

```
...
```

- 1 2 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.

Verification

To verify that a component has moved, you can use the **oc get pod -o wide** command.

For example:

- You want to move the Kibana pod from the **ip-10-0-147-79.us-east-2.compute.internal** node:

```
$ oc get pod kibana-5b8bdf44f9-ccpq9 -o wide
```

Example output

```
NAME                                READY STATUS  RESTARTS  AGE  IP             NODE
NOMINATED NODE  READINESS GATES
kibana-5b8bdf44f9-ccpq9  2/2   Running  0        27s  10.129.2.18  ip-10-0-147-79.us-east-2.compute.internal  <none>      <none>
```

- You want to move the Kibana pod to the **ip-10-0-139-48.us-east-2.compute.internal** node, a dedicated infrastructure node:

```
$ oc get nodes
```

Example output

```
NAME                                STATUS  ROLES    AGE  VERSION
ip-10-0-133-216.us-east-2.compute.internal  Ready  master   60m  v1.22.1
ip-10-0-139-146.us-east-2.compute.internal  Ready  master   60m  v1.22.1
ip-10-0-139-192.us-east-2.compute.internal  Ready  worker   51m  v1.22.1
ip-10-0-139-241.us-east-2.compute.internal  Ready  worker   51m  v1.22.1
ip-10-0-147-79.us-east-2.compute.internal  Ready  worker   51m  v1.22.1
ip-10-0-152-241.us-east-2.compute.internal  Ready  master   60m  v1.22.1
ip-10-0-139-48.us-east-2.compute.internal  Ready  infra    51m  v1.22.1
```

Note that the node has a **node-role.kubernetes.io/infra: "** label:

```
$ oc get node ip-10-0-139-48.us-east-2.compute.internal -o yaml
```

Example output

```
kind: Node
apiVersion: v1
metadata:
  name: ip-10-0-139-48.us-east-2.compute.internal
  selfLink: /api/v1/nodes/ip-10-0-139-48.us-east-2.compute.internal
```



```
uid: 62038aa9-661f-41d7-ba93-b5f1b6ef8751
resourceVersion: '39083'
creationTimestamp: '2020-04-13T19:07:55Z'
labels:
  node-role.kubernetes.io/infra: "
...
```

- To move the Kibana pod, edit the **ClusterLogging** CR to add a node selector:

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging

...

spec:

...

visualization:
  kibana:
    nodeSelector: 1
    node-role.kubernetes.io/infra: "
    proxy:
      resources: null
    replicas: 1
    resources: null
    type: kibana
```

- 1 Add a node selector to match the label in the node specification.

- After you save the CR, the current Kibana pod is terminated and new pod is deployed:

```
$ oc get pods
```

Example output

NAME	READY	STATUS	RESTARTS	AGE
cluster-logging-operator-84d98649c4-zb9g7	1/1	Running	0	29m
elasticsearch-cdm-hwv01pf7-1-56588f554f-kpmlg	2/2	Running	0	28m
elasticsearch-cdm-hwv01pf7-2-84c877d75d-75wqj	2/2	Running	0	28m
elasticsearch-cdm-hwv01pf7-3-f5d95b87b-4nx78	2/2	Running	0	28m
fluentd-42dzz	1/1	Running	0	28m
fluentd-d74rq	1/1	Running	0	28m
fluentd-m5vr9	1/1	Running	0	28m
fluentd-nkx17	1/1	Running	0	28m
fluentd-pdvqb	1/1	Running	0	28m
fluentd-tflh6	1/1	Running	0	28m
kibana-5b8bdf44f9-ccpq9	2/2	Terminating	0	4m11s
kibana-7d85dcffc8-bfpfp	2/2	Running	0	33s

- The new pod is on the **ip-10-0-139-48.us-east-2.compute.internal** node:

```
$ oc get pod kibana-7d85dcffc8-bfpfp -o wide
```

Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
NOMINATED NODE	READINESS	GATES				
kibana-7d85dcffc8-bfpfp	2/2	Running	0	43s	10.131.0.22	ip-10-0-139-48.us-east-2.compute.internal
	<none>	<none>				

- After a few moments, the original Kibana pod is removed.

```
$ oc get pods
```

Example output

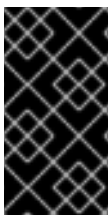
NAME	READY	STATUS	RESTARTS	AGE
cluster-logging-operator-84d98649c4-zb9g7	1/1	Running	0	30m
elasticsearch-cdm-hwv01pf7-1-56588f554f-kpmlg	2/2	Running	0	29m
elasticsearch-cdm-hwv01pf7-2-84c877d75d-75wqj	2/2	Running	0	29m
elasticsearch-cdm-hwv01pf7-3-f5d95b87b-4nx78	2/2	Running	0	29m
fluentd-42dzz	1/1	Running	0	29m
fluentd-d74rq	1/1	Running	0	29m
fluentd-m5vr9	1/1	Running	0	29m
fluentd-nkx17	1/1	Running	0	29m
fluentd-pdvqb	1/1	Running	0	29m
fluentd-tflh6	1/1	Running	0	29m
kibana-7d85dcffc8-bfpfp	2/2	Running	0	62s

4.7. 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 sum of CPU and memory requests of all pods running on the node is less than 50% of the allocated resources on the node.
- The cluster autoscaler can move all pods running on the node to the other 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.

4.7.1. ClusterAutoscaler 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 10
        min: 0 11
        max: 4 12
  scaleDown: 13
    enabled: true 14
    delayAfterAdd: 10m 15
    delayAfterDelete: 5m 16
    delayAfterFailure: 30s 17
    unneededTime: 5m 18
```

- 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 10 Optionally, specify the type of GPU node to deploy. Only **nvidia.com/gpu** and **amd.com/gpu** are valid types.
- 8 11 Specify the minimum number of GPUs to deploy in the cluster.
- 9 12 Specify the maximum number of GPUs to deploy in the cluster.
- 13 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**.
- 14 Specify whether the cluster autoscaler can remove unnecessary nodes.
- 15 Optionally, 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.
- 16 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 **10s** is used.
- 17 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.
- 18 Specify the period before an unnecessary node is eligible for deletion. If you do not specify a value, the default value of **10m** is used.



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.

4.7.2. Deploying the cluster autoscaler

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

Procedure

1. Create a YAML file for the **ClusterAutoscaler** resource that contains the customized resource definition.
2. Create the resource in the cluster:

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

- 1 **<filename>** is the name of the resource file that you customized.

4.8. ABOUT THE MACHINE AUTOSCALER

The machine autoscaler adjusts the number of Machines in the machine sets that you deploy in an OpenShift Container Platform cluster. You can scale both the default **worker** machine set and any other 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 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 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.

4.8.1. MachineAutoscaler 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 machine set this machine autoscaler scales, specify or include the name of the machine set to scale. The machine set name takes the following form: **<clusterid>-<machineset>-<aws-region-az>**.
- 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 machine set with extra large machines. The cluster autoscaler scales the 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 plane 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 AWS 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 machine set to scale.
- 5 The **kind** parameter value is always **MachineSet**.
- 6 The **name** value must match the name of an existing machine set, as shown in the **metadata.name** parameter value.

4.8.2. Deploying the machine autoscaler

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

Procedure

1. Create a YAML file for the **MachineAutoscaler** resource that contains the customized resource definition.
2. Create the resource in the cluster:

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

- 1 **<filename>** is the name of the resource file that you customized.

4.9. 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).

4.9.1. 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.

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

You can activate any of the following feature sets by using the **FeatureGate** CR:

Feature Set	Description
TechPreviewNoUpgrade	<p>Enables Technology Preview features that are not part of the default features. Enabling this feature set cannot be undone and prevents upgrades. This feature set is not recommended on production clusters.</p> <p>The following Technology Preview features are enabled by this feature set:</p> <ul style="list-style-type: none"> • RotateKubeletServerCertificate. Enables the rotation of the server TLS certificate on the kubelet. • Pod PID limits (SupportPodPidsLimit). Enables limiting process IDs (PIDs) in pods. • InsightsOperatorPullingSCA. Enables importing of RHEL Simple Content Access (SCA) certificates from Red Hat OpenShift Cluster Manager.

4.9.2. 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
```

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 sets that you want to enable in a comma-separated list:

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

**NOTE**

Enabling the **TechPreviewNoUpgrade** feature set cannot be undone and prevents upgrades. These feature sets are not recommended on production clusters.

4.10. ETCD TASKS

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

4.10.1. 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**

Keep in mind that etcd encryption only encrypts values, not keys. This means that resource types, namespaces, and object names are unencrypted.

4.10.2. Enabling etcd encryption

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

**WARNING**

It is not recommended to take a backup of etcd until the initial encryption process is complete. If the encryption process has not completed, the backup might be only partially encrypted.

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.

4.10.3. 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"}'
```

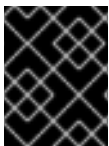
The output shows **DecryptionCompleted** upon successful decryption:

```
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.

4.10.4. 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 for a control plane node:

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

2. Change your root directory to the host:

```
sh-4.2# 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 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_<timestamp>.db**: This file is the etcd snapshot. The **cluster-backup.sh** script confirms its validity.
- **static_kuberesources_<timestamp>.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.

4.10.5. Defragmenting etcd data

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.

4.10.5.1. 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

Example log output

```
I0907 08:43:12.171919    1
defragcontroller.go:198] etcd member "ip-
10-0-191-150.us-west-2.compute.internal"
backend store fragmented: 39.33 %, dbSize:
349138944
```

4.10.5.2. Manual defragmentation

You can monitor the **etcd_db_total_size_in_bytes** metric to determine whether manual defragmentation is necessary.



WARNING

Defragmenting etcd is a blocking action. The etcd member will not response 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

- Determine which etcd member is the leader, because the leader should be defragmented last.
 - Get the list of etcd pods:

```
$ oc get pods -n openshift-etcd -o wide | grep -v quorum-guard | grep etcd
```

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>
```

- 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.us-west-1.compute.internal 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 |
| 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

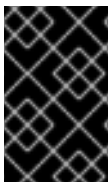
```
memberID:12345678912345678912 alarm:NOSPACE
```

- b. Clear the alarms:

```
sh-4.4# etcdctl alarm disarm
```

4.10.6. 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.



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

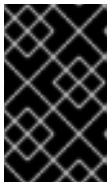
- Access to the cluster as a user with the **cluster-admin** role.
- A healthy control plane host to use as the recovery host.
- SSH access to control plane hosts.
- A backup directory containing both the etcd snapshot and the resources for the static pods, which were from the same backup. The file names in the directory must be in the following formats: **snapshot_<timestamp>.db** and **static_kuberesources_<timestamp>.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**

It is not required to manually stop the pods on the recovery host. The recovery script will stop the 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:

```
[core@ip-10-0-154-194 ~]$ sudo mv /etc/kubernetes/manifests/etcd-pod.yaml /tmp
```

- c. Verify that the etcd pods are stopped.

```
[core@ip-10-0-154-194 ~]$ sudo crictl ps | grep etcd | grep -v operator
```

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:

```
[core@ip-10-0-154-194 ~]$ sudo mv /etc/kubernetes/manifests/kube-apiserver-pod.yaml /tmp
```

- e. Verify that the Kubernetes API server pods are stopped.

■

```
[core@ip-10-0-154-194 ~]$ sudo crictl ps | grep kube-apiserver | grep -v operator
```

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:

```
[core@ip-10-0-154-194 ~]$ sudo mv /var/lib/etcd/ /tmp
```

- g. 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 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.

7. Run the restore script on the recovery control plane host and pass in the path to the etcd backup directory:

```
[core@ip-10-0-143-125 ~]$ sudo -E /usr/local/bin/cluster-restore.sh /home/core/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
```

8. Restart the kubelet service on all control plane hosts.

- a. From the recovery host, run the following command:

```
[core@ip-10-0-143-125 ~]$ sudo systemctl restart kubelet.service
```

- b. Repeat this step on all other control plane hosts.

9. Approve the pending CSRs:

- a. Get the list of current CSRs:

```
$ oc get csr
```

Example output

```
NAME      AGE  SIGNERNAME                                REQUESTOR
CONDITION
csr-2s94x  8m3s  kubernet.es.io/kubelet-serving            system:node:<node_name>
Pending 1
csr-4bd6t  8m3s  kubernet.es.io/kubelet-serving            system:node:<node_name>
Pending 2
csr-4hl85  13m   kubernet.es.io/kube-apiserver-client-kubelet
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper Pending
3
csr-zh4hp  3m8s  kubernet.es.io/kube-apiserver-client-kubelet
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper Pending
4
...
```

1 **1** **2** A pending kubelet service CSR (for user-provisioned installations).

3 **4** A pending **node-bootstrapper** CSR.

- b. Review the details of a CSR to verify that it is valid:

```
$ oc describe csr <csr_name> 1
```

1 **<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>
```

10. Verify that the single member control plane has started successfully.

- a. From the recovery host, verify that the etcd container is running.

```
[core@ip-10-0-143-125 ~]$ sudo crictl ps | grep etcd | grep -v operator
```

Example output

```
3ad41b7908e32
36f86e2eeaafe662df0d21041eb22b8198e0e58abeeae8c743c3e6e977e8009
About a minute ago   Running           etcd           0
7c05f8af362f0
```

- b. From the recovery host, verify that the etcd pod is running.

```
[core@ip-10-0-143-125 ~]$ oc get pods -n openshift-etcd | grep -v etcd-quorum-guard |
grep etcd
```



NOTE

If you attempt to run **oc login** prior to running this command and receive the following error, wait a few moments for the authentication controllers to start and try again.

```
Unable to connect to the server: EOF
```

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.

- c. Repeat this step for each lost control plane host that is not the recovery host.
11. Delete and recreate other non-recovery, control plane machines, one by one. After these machines are recreated, a new revision is forced and etcd scales up automatically. If you are running installer-provisioned infrastructure, or you used the Machine API to create your machines, follow these steps. Otherwise, you must create the new master node using the same method that was used to originally create it.



WARNING

Do not delete and recreate the machine for the recovery host.

- 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 NODE	PHASE PROVIDERID	TYPE	REGION STATE	ZONE	AGE
clustername-8qw5l-master-0 3h37m ip-10-0-131-183.ec2.internal	Running	m4.xlarge	us-east-1	us-east-1a	stopped
clustername-8qw5l-master-1 3h37m ip-10-0-143-125.ec2.internal	Running	m4.xlarge	us-east-1	us-east-1b	running
clustername-8qw5l-master-2 3h37m ip-10-0-154-194.ec2.internal	Running	m4.xlarge	us-east-1	us-east-1c	running
clustername-8qw5l-worker-us-east-1a-wbtgd 1a 3h28m ip-10-0-129-226.ec2.internal	Running	m4.large	us-east-1	us-east-1a	running
clustername-8qw5l-worker-us-east-1b-lrdxb 3h28m ip-10-0-144-248.ec2.internal	Running	m4.large	us-east-1	us-east-1b	running
clustername-8qw5l-worker-us-east-1c-pkg26 1c 3h28m ip-10-0-170-181.ec2.internal	Running	m4.large	us-east-1	us-east-1c	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. Update the **metadata.selfLink** field to use the new machine name from the previous step:

```
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  ...
  selfLink: /apis/machine.openshift.io/v1beta1/namespaces/openshift-machine-api/machines/clustername-8qw5l-master-3
  ...
```

- iv. Remove the **spec.providerID** field:

```
providerID: aws:///us-east-1a/i-0fdb85790d76d0c3f
```

- v. Remove the **metadata.annotations** and **metadata.generation** fields:

```
annotations:
  machine.openshift.io/instance-state: running
  ...
generation: 2
```

- vi. 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 NODE	PHASE PROVIDERID	TYPE	REGION STATE	ZONE	AGE
clustername-8qw5l-master-1 3h37m ip-10-0-143-125.ec2.internal	Running	m4.xlarge	us-east-1	us-east-1b	running
clustername-8qw5l-master-2 3h37m ip-10-0-154-194.ec2.internal	Running	m4.xlarge	us-east-1	us-east-1c	running
clustername-8qw5l-worker-us-east-1a-wbtgd 1a 3h28m ip-10-0-129-226.ec2.internal	Running	m4.large	us-east-1	us-east-1a	running
clustername-8qw5l-worker-us-east-1b-lrdxb 3h28m ip-10-0-144-248.ec2.internal	Running	m4.large	us-east-1	us-east-1b	running
clustername-8qw5l-worker-us-east-1c-pkg26 1c 3h28m ip-10-0-170-181.ec2.internal	Running	m4.large	us-east-1	us-east-1c	running

- f. Create the new machine 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 NODE	PHASE PROVIDERID	TYPE	REGION STATE	ZONE	AGE
clustername-8qw5l-master-1 3h37m ip-10-0-143-125.ec2.internal	Running	m4.xlarge	us-east-1	us-east-1b	running
clustername-8qw5l-master-2 3h37m ip-10-0-154-194.ec2.internal	Running	m4.xlarge	us-east-1	us-east-1c	running
clustername-8qw5l-master-3 85s ip-10-0-173-171.ec2.internal	Provisioning	m4.xlarge	us-east-1	us-east-1a	running
clustername-8qw5l-worker-us-east-1a-wbtgd east-1a 3h28m ip-10-0-129-226.ec2.internal	Running	m4.large	us-east-1	us-east-1a	running
clustername-8qw5l-worker-us-east-1b-lrdxb 1b 3h28m ip-10-0-144-248.ec2.internal	Running	m4.large	us-east-1	us-east-1b	running
clustername-8qw5l-worker-us-east-1c-pkg26 east-1c 3h28m ip-10-0-170-181.ec2.internal	Running	m4.large	us-east-1	us-east-1c	running

- 1 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.

12. In a separate terminal window, log in to the cluster as a user with the **cluster-admin** role by using the following command:

```
$ oc login -u <cluster_admin> 1
```

- 1 For **<cluster_admin>**, specify a user name with the **cluster-admin** role.

13. Force etcd redeployment.

In a terminal that has access to the cluster as a **cluster-admin** user, run the following command:

```
$ oc patch etcd cluster -p='{ "spec": { "forceRedeploymentReason": "recovery-"$( date --rfc-3339=ns )"' --type=merge 1
```

- 1 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.

14. 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.

15. 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 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.

- 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 **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.

16. 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 get pods -n openshift-etcd | grep -v etcd-quorum-guard | grep etcd
```

Example output

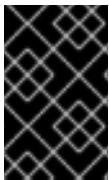
etcd-ip-10-0-143-125.ec2.internal	2/2	Running	0	9h
etcd-ip-10-0-154-194.ec2.internal	2/2	Running	0	9h
etcd-ip-10-0-173-171.ec2.internal	2/2	Running	0	9h

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 that it might take several minutes after completing this procedure for all services to be restored. For example, authentication by using **oc login** might not immediately work until the OAuth server pods are restarted.

4.10.7. 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*).

4.11. POD DISRUPTION BUDGETS

Understand and configure pod disruption budgets.

4.11.1. Understanding how to use pod disruption budgets to specify the number of pods that must be up

A *pod disruption budget* is part of the [Kubernetes](#) API, which can be managed with **oc** commands like other object types. They allow 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

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	SELECTOR
another-project	another-pdb	4	bar=foo
test-project	my-pdb	2	foo=bar

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.

4.11.2. 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/v1beta1 1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  minAvailable: 2 2
  selector: 3
    matchLabels:
      foo: bar
```

- 1** **PodDisruptionBudget** is part of the **policy/v1beta1** 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%**.
- 3** A label query over a set of resources. The result of **matchLabels** and **matchExpressions** are logically conjoined.

Or:

```
apiVersion: policy/v1beta1 1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  maxUnavailable: 25% 2
```

```
selector: ❸
matchLabels:
  foo: bar
```

- ❶ **PodDisruptionBudget** is part of the **policy/v1beta1** API group.
- ❷ 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%**.
- ❸ A label query over a set of resources. The result of **matchLabels** and **matchExpressions** are logically conjoined.

2. Run the following command to add the object to project:

```
$ oc create -f </path/to/file> -n <project_name>
```

4.12. 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.

4.12.1. 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, AWS, Azure, and GCP are supported.
 - For passthrough mode, AWS, Azure, 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.




NOTE

When rotating the credentials for an Azure cluster that is using mint mode, do not delete or replace the service principal that was used during installation. Instead, generate new Azure service principal client secrets and update the OpenShift Container Platform secrets accordingly.

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

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 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[].kind=="<provider_spec>") | .spec.secretRef'
```

Where **<provider_spec>** is the corresponding value for your cloud provider: **AWSProviderSpec** for AWS, **AzureProviderSpec** for Azure, or **GCPProviderSpec** for GCP.

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> -n <secret_namespace>
```

Where **<secret_name>** is the name of a secret and **<secret_namespace>** is 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.

- 7. To verify that the credentials have changed:
 - a. In the **Administrator** perspective of the web console, navigate to **Workloads → Secrets**.
 - b. Verify that the contents of the **Value** field or fields are different than the previously recorded information.

4.12.2. 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**

4.13. 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.

4.13.1. 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>**.

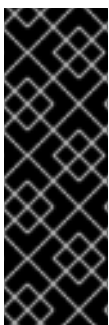
During a disconnected installation of OpenShift Container Platform, the status of the Cluster Samples Operator is set to **Removed**. If you choose to change it to **Managed**, it installs samples.

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.

4.13.2. 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.



IMPORTANT

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

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

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/rhsc/ruby-25-rhel7:latest ${MIRROR_ADDR}/rhsc/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.

4.13.3. 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)
```

Additional resources

- [About the Cloud Credential Operator](#)
- [Amazon Web Services \(AWS\) secret format](#)
- [Microsoft Azure secret format](#)
- [Google Cloud Platform \(GCP\) secret format](#)

CHAPTER 5. POST-INSTALLATION NODE TASKS

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

5.1. ADDING RHEL COMPUTE MACHINES TO AN OPENSHIFT CONTAINER PLATFORM CLUSTER

Understand and work with RHEL compute nodes.

5.1.1. About adding RHEL compute nodes to a cluster

In OpenShift Container Platform 4.9, you have the option of using Red Hat Enterprise Linux (RHEL) machines as compute machines, which are also known as worker machines, in your cluster if you use a user-provisioned infrastructure installation. You must use Red Hat Enterprise Linux CoreOS (RHCOS) machines for the control plane, or master, machines in your cluster.

As with all installations that use user-provisioned infrastructure, if you choose to use RHEL compute machines in your cluster, you take responsibility for all operating system life cycle management and maintenance, including performing system updates, applying patches, and completing all other required tasks.



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.



IMPORTANT

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.

5.1.2. System requirements for RHEL compute nodes

The Red Hat Enterprise Linux (RHEL) compute, or worker, 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 7.9](#) or [RHEL 8.4](#) with "Minimal" installation option.



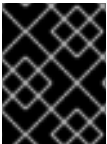
IMPORTANT

Adding RHEL 7 compute machines to an OpenShift Container Platform cluster is deprecated. Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.

In addition, you cannot upgrade your RHEL 7 compute machines 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 [Enabling FIPS Mode](#) in the RHEL 7 documentation.



IMPORTANT

The use of FIPS Validated / Modules in Process cryptographic libraries is only supported on OpenShift Container Platform deployments on the **x86_64** architecture.

- 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](#)

5.1.2.1. 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.

5.1.3. 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 4.9 cluster, you must prepare a RHEL 7 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 7 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 4.9:

```
# subscription-manager repos \
  --enable="rhel-7-server-rpms" \
  --enable="rhel-7-server-extras-rpms" \
  --enable="rhel-7-server-ansible-2.9-rpms" \
  --enable="rhel-7-server-ose-4.9-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.

5.1.4. 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.

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 4.9.

- a. For RHEL 7 nodes, you must enable the following repositories:

```
# subscription-manager repos \
  --enable="rhel-7-server-rpms" \
  --enable="rhel-7-fast-datapath-rpms" \
  --enable="rhel-7-server-extras-rpms" \
  --enable="rhel-7-server-optional-rpms" \
  --enable="rhel-7-server-ose-4.9-rpms"
```



NOTE

Use of RHEL 7 nodes is deprecated and planned for removal in a future release of OpenShift Container Platform 4.

- b. For RHEL 8 nodes, you must enable the following repositories:

```
# subscription-manager repos \
  --enable="rhel-8-for-x86_64-baseos-rpms" \
  --enable="rhel-8-for-x86_64-appstream-rpms" \
  --enable="rhocp-4.9-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.

5.1.5. 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 4.9 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 ❶
#ansible_become=True ❷

openshift_kubeconfig_path=~/.kube/config" ❸

[new_workers] ❹
mycluster-rhel8-0.example.com
mycluster-rhel8-1.example.com
```

- ❶ Specify the user name that runs the Ansible tasks on the remote compute machines.
- ❷ If you do not specify **root** for the **ansible_user**, you must set **ansible_become** to **True** and assign the user sudo permissions.
- ❸ Specify the path and file name of the **kubeconfig** file for your cluster.
- ❹ 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 ❶
```

- ❶ For **<path>**, specify the path to the Ansible inventory file that you created.

5.1.6. 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.

Paramter	Description	Values
----------	-------------	--------

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.

5.1.7. 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

- View the list of machines and record the node names of the RHCOS compute machines:

```
$ oc get nodes -o wide
```

- For each RHCOS compute machine, delete the node:

- Mark the node as unschedulable by running the **oc adm cordon** command:

```
$ oc adm cordon <node_name> 1
```

- Specify the node name of one of the RHCOS compute machines.

- Drain all the pods from the node:

```
$ oc adm drain <node_name> --force --delete-local-data --ignore-daemonsets 1
```

- Specify the node name of the RHCOS compute machine that you isolated.

- 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.

5.2. ADDING RHCOS COMPUTE MACHINES TO AN OPENSIFT 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.

5.2.1. 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](#).

5.2.2. 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. After the instance boots, press the **TAB** or **E** key to edit the kernel command line.
3. Add the parameters to the kernel command line:

```
coreos.inst.install_dev=sda 1
coreos.inst.ignition_url=http://example.com/worker.ign 2
```

- 1** Specify the block device of the system to install to.
 - 2** Specify the URL of the compute Ignition config file. Only HTTP and HTTPS protocols are supported.
4. Press **Enter** to complete the installation. After RHCOS installs, the system reboots. After the system reboots, it applies the Ignition config file that you specified.
 5. Continue to create more compute machines for your cluster.

5.2.3. 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.

5.2.4. 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	v1.22.1
master-1	Ready	master	63m	v1.22.1
master-2	Ready	master	64m	v1.22.1

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-bootstraptrapper	Pending
csr-8vnps	15m	system:serviceaccount:openshift-machine-config-operator:node-bootstraptrapper	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-bootstrap** 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> 1
```

- 1** **<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.

- 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
...
```

- 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> 1
```

- 1** **<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   v1.22.1
master-1  Ready    master   73m   v1.22.1
master-2  Ready    master   74m   v1.22.1
worker-0  Ready    worker   11m   v1.22.1
worker-1  Ready    worker   11m   v1.22.1
```



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](#).

5.3. DEPLOYING MACHINE HEALTH CHECKS

Understand and deploy machine health checks.



IMPORTANT

This process is not applicable for clusters with manually provisioned machines. You can use the advanced machine management and scaling capabilities only in clusters where the Machine API is operational.

5.3.1. About machine health checks

Machine health checks automatically repair unhealthy machines in a particular machine pool.

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.



NOTE

You cannot apply a machine health check to a machine with the master role.

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.

5.3.1.1. 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.
- Control plane machines are not currently supported and are not remediated if they are unhealthy.
- 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**.

5.3.2. 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.

5.3.2.1. 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 machine sets across multiple availability zones so that if you lose an entire zone, your **maxUnhealthy** setting prevents further remediation within the cluster.

The **maxUnhealthy** field can be set as either an integer or percentage. There are different remediation implementations depending on the **maxUnhealthy** value.

5.3.2.1.1. 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.

5.3.2.1.2. 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.

5.3.3. Creating a **MachineHealthCheck** resource

You can create a **MachineHealthCheck** resource for all **MachineSets** in your cluster. You should not create a **MachineHealthCheck** resource that targets control plane machines.

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
```

5.3.4. Scaling a machine set manually

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

This guidance is relevant to fully automated, installer-provisioned infrastructure installations. Customized, user-provisioned infrastructure installations do not have 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 machine sets that are in the cluster:

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

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

2. Scale the machine set:

```
$ 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 machine set:

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

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

5.3.5. Understanding the difference between 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.

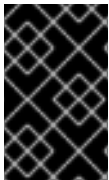
The **NodeSelector** object can be replaced with a reference to the **MachineSet** object.

5.4. 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.

podsPerCore 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**. **podsPerCore** cannot exceed **maxPods**.

maxPods sets the number of pods the node can run to a fixed value, regardless of the properties of the node.

```
kubeletConfig:
  maxPods: 250
```

5.4.1. 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:

- Create one **KubeletConfig** 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 of the pools, you need only one **KubeletConfig** CR for all of the pools.
- 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.
- 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**.

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:

```

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

```

- ❶ Enter the label from the machine config pool.
- ❷ 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=large-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
```

5.4.2. 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. Set **maxUnavailable** to the value that you want:

```
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.

5.4.3. Control plane node sizing

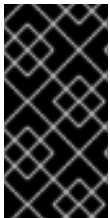
The control plane node resource requirements depend on the number of nodes in the cluster. The following control plane node size recommendations are based on the results of control plane density focused testing. The control plane tests create the following objects across the cluster in each of the namespaces depending on the node counts:

- 12 image streams
- 3 build configurations
- 6 builds
- 1 deployment with 2 pod replicas mounting two secrets each
- 2 deployments with 1 pod replica mounting two secrets
- 3 services pointing to the previous deployments
- 3 routes pointing to the previous deployments
- 10 secrets, 2 of which are mounted by the previous deployments
- 10 config maps, 2 of which are mounted by the previous deployments

Number of worker nodes	Cluster load (namespaces)	CPU cores	Memory (GB)
25	500	4	16

Number of worker nodes	Cluster load (namespaces)	CPU cores	Memory (GB)
100	1000	8	32
250	4000	16	96

On a large and dense cluster with three masters or 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 or underlying infrastructure in addition to 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 masters 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 half 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 it's 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
2000	2	4.4
3000	2.7	5.6
4000	3.8	7.6
5000	4.2	9.02
6000	5.8	11.3

Number of namespaces	OLM memory at idle state (GB)	OLM memory with 5 user operators installed (GB)
7000	6.6	12.9
8000	6.9	14.8
9000	8	17.7
10,000	9.9	21.6

**IMPORTANT**

If you used an installer-provisioned infrastructure installation method, you cannot modify the control plane node size in a running OpenShift Container Platform 4.9 cluster. Instead, 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 plug-in.

**NOTE**

In OpenShift Container Platform 4.9, 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.

5.4.4. Setting up CPU Manager

Procedure

- Optional: Label a node:

```
# oc label node perf-node.example.com cpumanager=true
```

- 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
```

- 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 1
    cpuManagerReconcilePeriod: 5s 2
```

1

Specify a policy:

- **none**. This policy explicitly enables the existing default CPU affinity scheme, providing no affinity beyond what the scheduler does automatically.
- **static**. This policy allows pods with certain resource characteristics to be granted increased CPU affinity and exclusivity on the node.

2Optional. 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 ² These settings were defined when you created 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

```

5.5. HUGE PAGES

Understand and configure huge pages.

5.5.1. 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.

5.5.2. 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"
        requests:
          hugepages-2Mi: 100Mi
      volumeName: hugepage
  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*.

Additional resources

- [Configuring Transparent Huge Pages](#)

5.5.3. Configuring huge pages

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.

5.5.3.1. At boot time

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 3
      name: openshift-node-hugepages

  recommend:
    - machineConfigLabels: 4
      machineconfiguration.openshift.io/role: "worker-hp"
    priority: 30
    profile: openshift-node-hugepages
```

- 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
```



WARNING

This functionality is currently only supported on Red Hat Enterprise Linux CoreOS (RHCOS) 8.x worker nodes. On Red Hat Enterprise Linux (RHEL) 7.x worker nodes the Tuned **[bootloader]** plug-in is currently not supported.

5.6. UNDERSTANDING DEVICE PLUG-INS

The device plug-in provides a consistent and portable solution to consume hardware devices across

clusters. The device plug-in 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 plug-in API, but the device plug-in Containers are supported by individual vendors.

A device plug-in is a gRPC service running on the nodes (external to the **kubelet**) that is responsible for managing specific hardware resources. Any device plug-in 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 plug-ins

- [Nvidia GPU device plug-in for COS-based operating system](#)
- [Nvidia official GPU device plug-in](#)
- [Solarflare device plug-in](#)
- [KubeVirt device plug-ins: vfio and kvm](#)
- [Kubernetes device plug-in for IBM Crypto Express \(CEX\) cards](#)



NOTE

For easy device plug-in reference implementation, there is a stub device plug-in in the Device Manager code:
[vendor/k8s.io/kubernetes/pkg/kubelet/cm/deviceplugin/device_plugin_stub.go](https://github.com/kubernetes/kubernetes/blob/master/vendor/k8s.io/kubernetes/pkg/kubelet/cm/deviceplugin/device_plugin_stub.go).

5.6.1. Methods for deploying a device plug-in

- Daemon sets are the recommended approach for device plug-in deployments.
- Upon start, the device plug-in 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 plug-ins 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 plug-in implementation.

5.6.2. Understanding the Device Manager

Device Manager provides a mechanism for advertising specialized node hardware resources with the help of plug-ins known as device plug-ins.

You can advertise specialized hardware without requiring any upstream code changes.



IMPORTANT

OpenShift Container Platform supports the device plug-in API, but the device plug-in 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 plug-in 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 plug-in service. In response, Device Manager gets a list of **Device** objects from the plug-in over a gRPC stream. Device Manager will keep watching on the stream for new updates from the plug-in. On the plug-in side, the plug-in 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 plug-in exists or not. If the plug-in exists and there are free allocatable devices as well as per local cache, **Allocate** RPC is invoked at that particular device plug-in.

Additionally, device plug-ins can also perform several other device-specific operations, such as driver installation, device initialization, and device resets. These functionalities vary from implementation to implementation.

5.6.3. Enabling Device Manager

Enable Device Manager to implement a device plug-in to advertise specialized hardware without any upstream code changes.

Device Manager provides a mechanism for advertising specialized node hardware resources with the help of plug-ins known as device plug-ins.

1. Obtain the label associated with the static **MachineConfigPool** CRD for the type of node you want to configure. 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 plug-in registrations. This sock file is created when the Kubelet is started only if Device Manager is enabled.

5.7. TAINTS AND TOLERATIONS

Understand and work with taints and tolerations.

5.7.1. 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

```
spec:
  ...
  template:
    ...
    spec:
      taints:
        - effect: NoExecute
          key: key1
          value: value1
      ...
```

Example toleration in a Pod spec

```
spec:
  ...
  template:
    ...
    spec:
      tolerations:
        - key: "key1"
          operator: "Equal"
          value: "value1"
          effect: "NoExecute"
          tolerationSeconds: 3600
      ...
```

Taints and tolerations consist of a key, value, and effect.

Table 5.1. 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.

Parameter	Description						
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	<p>The effect is one of the following:</p> <table> <tr> <td>NoSchedule ^[1]</td><td> <ul style="list-style-type: none"> 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> <ul style="list-style-type: none"> 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> <ul style="list-style-type: none"> 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>	NoSchedule ^[1]	<ul style="list-style-type: none"> New pods that do not match the taint are not scheduled onto that node. Existing pods on the node remain. 	PreferNoSchedule	<ul style="list-style-type: none"> 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. 	NoExecute	<ul style="list-style-type: none"> 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.
NoSchedule ^[1]	<ul style="list-style-type: none"> New pods that do not match the taint are not scheduled onto that node. Existing pods on the node remain. 						
PreferNoSchedule	<ul style="list-style-type: none"> 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. 						
NoExecute	<ul style="list-style-type: none"> 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. 						
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>	Equal	The key/value/effect parameters must match. This is the default.	Exists	The key/effect parameters must match. You must leave a blank value parameter, which matches any.		
Equal	The key/value/effect parameters must match. This is the default.						
Exists	The key/effect parameters must match. You must leave a blank value parameter, which matches any.						

- 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
...
spec:
  taints:
```



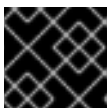
```
- effect: NoSchedule
  key: node-role.kubernetes.io/master
  ...
```

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**.

5.7.1.1. Understanding how to use toleration seconds to delay pod evictions

You can specify how long a pod can remain bound to a node before being evicted by specifying the **tolerationSeconds** parameter in the **Pod** specification or **MachineSet** object. If a taint with the **NoExecute** effect is added to a node, a pod that does tolerate the taint, which has the **tolerationSeconds** parameter, the pod is not evicted until that time period expires.

Example output

```
spec:
....
  template:
....
    spec:
      tolerations:
      - key: "key1"
        operator: "Equal"
        value: "value1"
        effect: "NoExecute"
        tolerationSeconds: 3600
```

Here, if this pod is running but does not have a matching toleration, the pod stays bound to the node for 3,600 seconds and then be evicted. If the taint is removed before that time, the pod is not evicted.

5.7.1.2. Understanding how to use multiple taints

You can put multiple taints on the same node and multiple tolerations on the same pod. OpenShift Container Platform processes multiple taints and tolerations as follows:

1. Process the taints for which the pod has a matching toleration.
2. The remaining unmatched taints have the indicated effects on the pod:
 - If there is at least one unmatched taint with effect **NoSchedule**, OpenShift Container Platform cannot schedule a pod onto that node.
 - If there is no unmatched taint with effect **NoSchedule** but there is at least one unmatched taint with effect **PreferNoSchedule**, OpenShift Container Platform tries to not schedule the pod onto the node.
 - If there is at least one unmatched taint with effect **NoExecute**, OpenShift Container Platform evicts the pod from the node if it is already running on the node, or the pod is not scheduled onto the node if it is not yet running on the node.
 - Pods that do not tolerate the taint are evicted immediately.
 - Pods that tolerate the taint without specifying **tolerationSeconds** in their **Pod** specification remain bound forever.
 - Pods that tolerate the taint with a specified **tolerationSeconds** remain bound for the specified amount of time.

For example:

- Add the following taints to the node:

```
$ oc adm taint nodes node1 key1=value1:NoSchedule
```

```
$ oc adm taint nodes node1 key1=value1:NoExecute
```

```
$ oc adm taint nodes node1 key2=value2:NoSchedule
```

- The pod has the following tolerations:

```

spec:
  ....
  template:
    ....
    spec:
      tolerations:
        - key: "key1"
          operator: "Equal"
          value: "value1"
          effect: "NoSchedule"
        - key: "key1"
          operator: "Equal"
          value: "value1"
          effect: "NoExecute"

```

In this case, the pod cannot be scheduled onto the node, because there is no toleration matching the third taint. The pod continues running if it is already running on the node when the taint is added, because the third taint is the only one of the three that is not tolerated by the pod.

5.7.1.3. Understanding pod scheduling and node conditions (taint node by condition)

The Taint Nodes By Condition feature, which is enabled by default, automatically taints nodes that report conditions such as memory pressure and disk pressure. If a node reports a condition, a taint is added until the condition clears. The taints have the **NoSchedule** effect, which means no pod can be scheduled on the node unless the pod has a matching toleration.

The scheduler checks for these taints on nodes before scheduling pods. If the taint is present, the pod is scheduled on a different node. Because the scheduler checks for taints and not the actual node conditions, you configure the scheduler to ignore some of these node conditions by adding appropriate pod tolerations.

To ensure backward compatibility, the daemon set controller automatically adds the following tolerations to all daemons:

- node.kubernetes.io/memory-pressure
- node.kubernetes.io/disk-pressure
- node.kubernetes.io/unschedulable (1.10 or later)
- node.kubernetes.io/network-unavailable (host network only)

You can also add arbitrary tolerations to daemon sets.

5.7.1.4. Understanding evicting pods by condition (taint-based evictions)

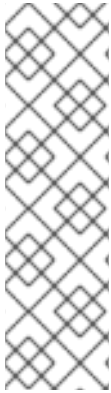
The Taint-Based Evictions feature, which is enabled by default, evicts pods from a node that experiences specific conditions, such as **not-ready** and **unreachable**. When a node experiences one of these conditions, OpenShift Container Platform automatically adds taints to the node, and starts evicting and rescheduling the pods on different nodes.

Taint Based Evictions have a **NoExecute** effect, where any pod that does not tolerate the taint is evicted immediately and any pod that does tolerate the taint will never be evicted, unless the pod uses the **tolerationSeconds** parameter.

The **tolerationSeconds** parameter allows you to specify how long a pod stays bound to a node that has

a node condition. If the condition still exists after the **tolerationSeconds** period, the taint remains on the node and the pods with a matching toleration are evicted. If the condition clears before the **tolerationSeconds** period, pods with matching tolerations are not removed.

If you use the **tolerationSeconds** parameter with no value, pods are never evicted because of the not ready and unreachable node conditions.



NOTE

OpenShift Container Platform evicts pods in a rate-limited way to prevent massive pod evictions in scenarios such as the master becoming partitioned from the nodes.

By default, if more than 55% of nodes in a given zone are unhealthy, the node lifecycle controller changes that zone's state to **PartialDisruption** and the rate of pod evictions is reduced. For small clusters (by default, 50 nodes or less) in this state, nodes in this zone are not tainted and evictions are stopped.

For more information, see [Rate limits on eviction](#) in the Kubernetes documentation.

OpenShift Container Platform automatically adds a toleration for **node.kubernetes.io/not-ready** and **node.kubernetes.io/unreachable** with **tolerationSeconds=300**, unless the **Pod** configuration specifies either toleration.

```
spec:
  ....
  template:
    ....
    spec:
      tolerations:
        - key: node.kubernetes.io/not-ready
          operator: Exists
          effect: NoExecute
          tolerationSeconds: 300 1
        - key: node.kubernetes.io/unreachable
          operator: Exists
          effect: NoExecute
          tolerationSeconds: 300
```

1 These tolerations ensure that the default pod behavior is to remain bound for five minutes after one of these node conditions problems is detected.

You can configure these tolerations as needed. For example, if you have an application with a lot of local state, you might want to keep the pods bound to node for a longer time in the event of network partition, allowing for the partition to recover and avoiding pod eviction.

Pods spawned by a daemon set are created with **NoExecute** tolerations for the following taints with no **tolerationSeconds**:

- **node.kubernetes.io/unreachable**
- **node.kubernetes.io/not-ready**

As a result, daemon set pods are never evicted because of these node conditions.

5.7.1.5. Tolerating all taints

You can configure a pod to tolerate all taints by adding an **operator: "Exists"** toleration with no **key** and **value** parameters. Pods with this toleration are not removed from a node that has taints.

Pod spec for tolerating all taints

```
spec:
  ....
  template:
    ....
    spec:
      tolerations:
        - operator: "Exists"
```

5.7.2. 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

```
spec:
  ....
  template:
    ....
    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

```
spec:
  ....
  template:
    ....
```

```
spec:
  tolerations:
  - key: "key1"
    operator: "Exists" ❶
    effect: "NoExecute"
    tolerationSeconds: 3600
```

- ❶ 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
...
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**.

5.7.3. Adding taints and tolerations using a machine set

You can add taints to nodes using a machine set. All nodes associated with the **MachineSet** object are updated with the taint. Tolerations respond to taints added by a 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

```
spec:
  ....
  template:
    ....
    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 is bound to a node before being evicted.

For example:

Sample pod configuration file with **Exists** operator

```
spec:
  ....
  template:
    ....
    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 node specification

```
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 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 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 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.

5.7.4. 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: <node_name>
  labels:
    ...
spec:
  taints:
    - key: dedicated
      value: groupName
      effect: NoSchedule
```

2. Add a toleration to the pods by writing a custom admission controller.

5.7.5. 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:

```
spec:
  ....
  template:
    ....
    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: <node_name>
  labels:
    ...
spec:
  taints:
    - key: disktype
      value: ssd
      effect: PreferNoSchedule
```

5.7.6. 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:

```
spec:
  ....
  template:
    ....
    spec:
      tolerations:
        - key: "key2"
```

```
operator: "Exists"
effect: "NoExecute"
tolerationSeconds: 3600
```

5.8. TOPOLOGY MANAGER

Understand and work with Topology Manager.

5.8.1. 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.



NOTE

To align CPU resources with other requested resources in a **Pod** spec, the CPU Manager must be enabled with the **static** CPU Manager policy.

Topology Manager supports four allocation policies, which you assign in the **cpumanager-enabled** custom resource (CR):

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.

5.8.2. Setting up Topology Manager

To use Topology Manager, you must configure an allocation policy in the **cpumanager-enabled** custom resource (CR). 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**. See the Using CPU Manager in the Scalability and Performance section.

Procedure

To activate Topology Manager:

1. Configure the Topology Manager allocation policy in the **cpumanager-enabled** custom resource (CR).

```
$ 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 ❶
    cpuManagerReconcilePeriod: 5s
    topologyManagerPolicy: single-numa-node ❷
```

- ❶ This parameter must be **static**.
- ❷ Specify your selected Topology Manager allocation policy. Here, the policy is **single-numa-node**. Acceptable values are: **default**, **best-effort**, **restricted**, **single-numa-node**.

5.8.3. 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 consults the CPU Manager static policy, which returns the topology of available CPUs. Topology Manager also consults Device Manager to discover the topology of available devices for example.com/device.

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.

5.9. 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.

5.10. CLUSTER-LEVEL OVERCOMMIT USING THE CLUSTER RESOURCE OVERRIDE OPERATOR

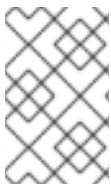
The Cluster Resource Override Operator is an admission webhook that allows you to control the level of overcommit and manage container density across all the nodes in your cluster. The Operator controls how nodes in specific projects can exceed defined memory and CPU limits.

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 2
      cpuRequestToLimitPercent: 25 3
      limitCPUMemoryPercent: 200 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.

5.10.1. 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 Instance**.
 - b. On the **Create ClusterResourceOverride** page, 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

c. Click **Create**.

4. Check the current state of the admission webhook by checking the status of the cluster custom resource:
- On the **ClusterResourceOverride Operator** page, click **cluster**.
 - 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:
    kubectrl.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/v1beta1
  kind: MutatingWebhookConfiguration
  name: clusterresourceoverrides.admission.autoscaling.openshift.io
  resourceVersion: "127621"
  uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

....
```

- 1 Reference to the **ClusterResourceOverride** admission webhook.

5.10.2. 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: "4.9"
  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","metadat
a":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":{"spec":
{"cpuRequestToLimitPercent":25,"limitCPUToMemoryPercent":200,"memoryRequestToLimitPe
rcent":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/v1beta1
  kind: MutatingWebhookConfiguration
  name: clusterresourceoverrides.admission.autoscaling.openshift.io
  resourceVersion: "127621"
  uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

....
```

- 1 Reference to the **ClusterResourceOverride** admission webhook.

5.10.3. 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 1
      cpuRequestToLimitPercent: 25 2
      limitCPUToMemoryPercent: 200 3
```

- 1 Optional. Specify the percentage to override the container memory limit, if used, between 1-100. The default is 50.
- 2 Optional. Specify the percentage to override the container CPU limit, if used, between 1-100. The default is 25.
- 3 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" 1
  ...
```

- 1 Add this label to each project.

5.11. 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.

5.11.1. Understanding compute resources and containers

The node-enforced behavior for compute resources is specific to the resource type.

5.11.1.1. 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.

5.11.1.2. 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.

5.11.2. Understanding overcommitment 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.

For each compute resource, a container is divided into one of three QoS classes with decreasing order of priority:

Table 5.2. 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 container 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 container 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 container 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.

5.11.2.1. 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 QoS 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.

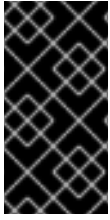
5.11.3. Understanding swap memory and QOS

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.

5.11.4. 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
```

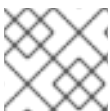
Example output

```
vm.overcommit_memory = 1
```

```
$ 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

5.11.5. 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

1. Obtain the label associated with the static **MachineConfigPool** CRD 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: small-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=small-pods
```


TIP

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

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  labels:
    custom-kubelet: small-pods
  name: worker
```

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 1
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: small-pods 2
  kubeletConfig:
    cpuCfsQuota: 3
      - "false"
```

- 1** Assign a name to CR.
- 2** Specify the label to apply the configuration change.
- 3** Set the **cpuCfsQuota** parameter to **false**.

5.11.6. 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](#).

5.11.7. 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
```

5.12. 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.

5.12.1. 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 project object file
2. Add the following annotation:

```
quota.openshift.io/cluster-resource-override-enabled: "false"
```

3. Create the project object:

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

5.13. FREEING NODE RESOURCES USING GARBAGE COLLECTION

Understand and use garbage collection.

5.13.1. Understanding how terminated containers are removed through garbage collection

Container garbage collection can be performed 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 5.3. Variables for configuring container garbage collection

Node condition	Eviction signal	Description
MemoryPressure	memory.available	The available memory on the node.
DiskPressure	<ul style="list-style-type: none"> • 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 .

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.

5.13.2. Understanding how images are removed through garbage collection

Image garbage collection relies on disk usage as reported by **cAdvisor** on the node to decide which images to remove from the node.

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 5.4. 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 .
imageGCLowThresholdPercent	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.

5.13.3. 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

Prerequisites

1. Obtain the label associated with the static **MachineConfigPool** CRD 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

```
Name:      worker
Namespace:
Labels:    custom-kubelet=small-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=small-pods
```

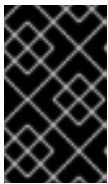
TIP

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

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  labels:
    custom-kubelet: small-pods
  name: worker
```

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 1
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: small-pods 2
  kubeletConfig:
    evictionSoft: 3
      memory.available: "500Mi" 4
      nodefs.available: "10%"
      nodefs.inodesFree: "5%"
      imagefs.available: "15%"
      imagefs.inodesFree: "10%"
    evictionSoftGracePeriod: 5
      memory.available: "1m30s"
      nodefs.available: "1m30s"
      nodefs.inodesFree: "1m30s"
      imagefs.available: "1m30s"
      imagefs.inodesFree: "1m30s"
    evictionHard:
      memory.available: "200Mi"
      nodefs.available: "5%"
      nodefs.inodesFree: "4%"
      imagefs.available: "10%"
      imagefs.inodesFree: "5%"
    evictionPressureTransitionPeriod: 0s 6
```

```
imageMinimumGCAge: 5m 7
imageGCHighThresholdPercent: 80 8
imageGCLowThresholdPercent: 75 9
```

- 1 Name for the object.
- 2 Selector label.
- 3 Type of eviction: **EvictionSoft** and **EvictionHard**.
- 4 Eviction thresholds based on a specific eviction trigger signal.
- 5 Grace periods for the soft eviction. This parameter does not apply to **eviction-hard**.
- 6 The duration to wait before transitioning out of an eviction pressure condition.
- 7 The minimum age for an unused image before the image is removed by garbage collection.
- 8 The percent of disk usage (expressed as an integer) which triggers image garbage collection.
- 9 The percent of disk usage (expressed as an integer) to which image garbage collection attempts to free.

2. Create the object:

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

For example:

```
$ oc create -f gc-container.yaml
```

Example output

```
kubeletconfig.machineconfiguration.openshift.io/gc-container created
```

3. Verify that garbage collection is active. 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

5.14. 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.

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 is part of a standard OpenShift Container Platform installation in version 4.1 and later.

5.14.1. Accessing an example Node Tuning Operator specification

Use this process to access an example Node Tuning Operator specification.

Procedure

1. Run:

```
$ oc get Tuned/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 might be deprecated in future versions of the Node Tuning Operator.

5.14.2. 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
```


■

- 1 Optional.
- 2 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**.

<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.

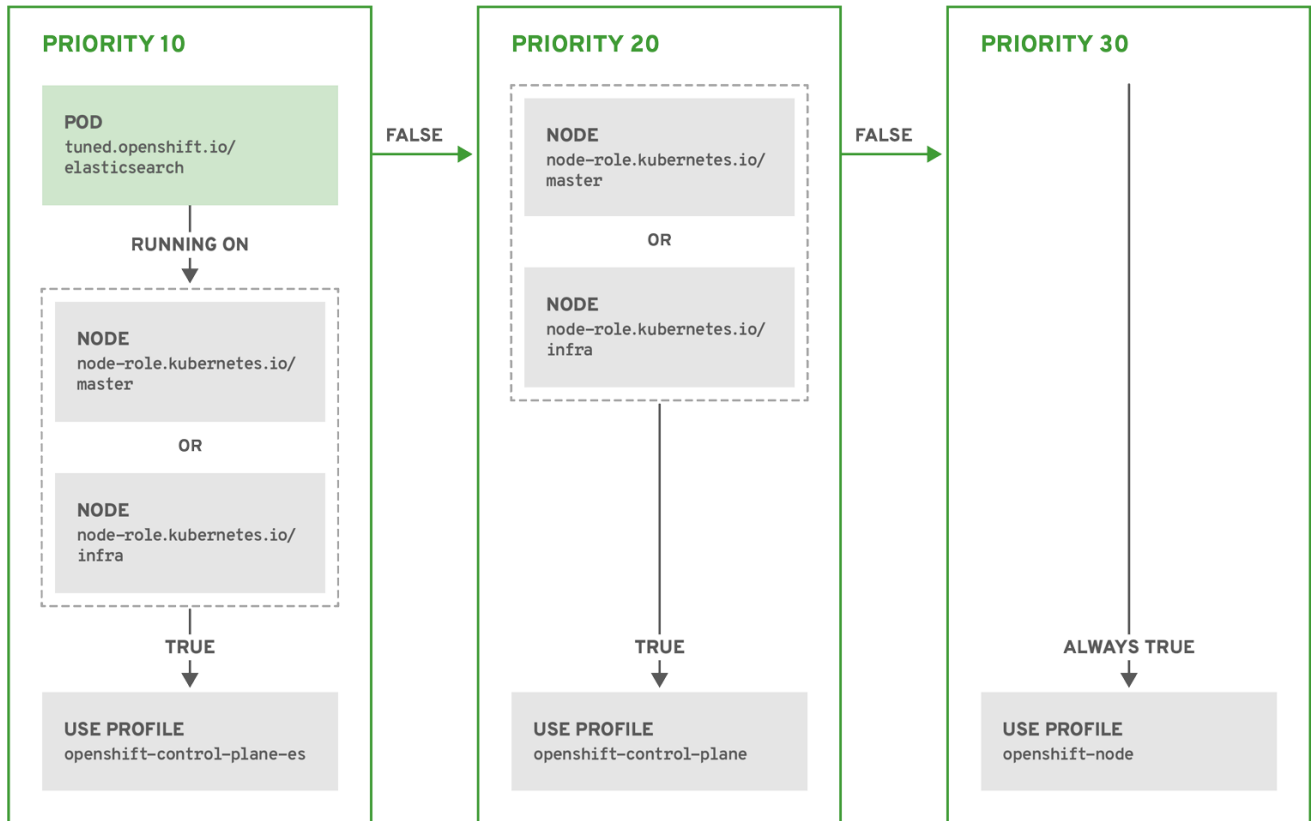
Example: node or pod label based matching

```
- 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.



OPENSIFT_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.

5.14.3. 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:
  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 ^ {} \;
```

5.14.4. Supported Tuned daemon plug-ins

Excluding the **[main]** section, the following Tuned plug-ins 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

There is some dynamic tuning functionality provided by some of these plug-ins that is not supported. The following TuneD plug-ins are currently not supported:

- bootloader
- script
- systemd

See [Available TuneD Plug-ins](#) and [Getting Started with TuneD](#) for more information.

5.15. 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. 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: small-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=small-pods
```

TIP

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

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  labels:
    custom-kubelet: small-pods
  name: worker
```

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:
      custom-kubelet: small-pods 2
  kubeletConfig:
    podsPerCore: 10 3
    maxPods: 250 4
```

- 1** Assign a name to CR.
- 2** Specify the label to apply the configuration change.
- 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. 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 6. POST-INSTALLATION NETWORK CONFIGURATION

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

6.1. 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

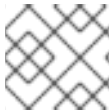
IP address pools from which pod IP addresses are allocated.

serviceNetwork

IP address pool for services.

defaultNetwork.type

Cluster network provider, such as OpenShift SDN or OVN-Kubernetes.



NOTE

After cluster installation, you cannot modify the fields listed in the previous section.

6.2. 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 ConfigMap 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: | 1
    <MY_PEM_ENCODED_CERTS> 2
kind: ConfigMap
metadata:
  name: user-ca-bundle 3
  namespace: openshift-config 4
```

- 1** This data key must be named **ca-bundle.crt**.
- 2** One or more PEM-encoded X.509 certificates used to sign the proxy's identity certificate.
- 3** The ConfigMap name that will be referenced from the Proxy object.
- 4** The ConfigMap must be in the **openshift-config** namespace.

- b. Create the ConfigMap 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> 1
  httpsProxy: http://<username>:<pswd>@<ip>:<port> 2
```

```
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.
- 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 ConfigMap in the **openshift-config** namespace that contains additional CA certificates required for proxying HTTPS connections. Note that the ConfigMap 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.

6.3. 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>-wfp4: owned
status: {}

```

6.4. 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.

6.5. CONFIGURING THE NODE PORT SERVICE RANGE

As a cluster administrator, you can expand the available node port range. If your cluster uses 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.

6.5.1. 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.

6.5.1.1. 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"]
```

6.6. CONFIGURING NETWORK POLICY

As a cluster administrator or project administrator, you can configure network policies for a project.

6.6.1. About network policy

In a cluster using a Kubernetes Container Network Interface (CNI) plug-in that supports Kubernetes network policy, network isolation is controlled entirely by **NetworkPolicy** objects. In OpenShift Container Platform 4.9, OpenShift SDN supports using network policy in its default network isolation mode.



NOTE

When using the OpenShift SDN cluster network provider, the following limitations apply regarding network policies:

- Egress network policy as specified by the **egress** field is not supported.
- IPBlock is supported by network policy, but without support for **except** clauses. If you create a policy with an IPBlock section that includes an **except** clause, the SDN pods log warnings and the entire IPBlock section of that policy is ignored.

**WARNING**

Network policy does not apply to the host network namespace. Pods with host networking enabled are unaffected by network policy rules.

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.

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.

6.6.2. 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 describing the pods the policy applies to. The policy object can only select pods in the project that the NetworkPolicy object is defined.
- 3** A selector matching the pods that the policy object allows ingress traffic from. The selector will match pods in any project.
- 4** A list of one or more destination ports to accept traffic on.

6.6.3. Creating a network policy

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 cluster network provider that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network provider or 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.
- Your cluster is using a cluster 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.

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

```
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: {}
```

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/default-deny created
```

**NOTE**

If you log in with a user with the **cluster-admin** role in the console, then you have a choice of creating a network policy in any namespace in the cluster directly from the YAML view or from a form in the web console.

6.6.4. 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 cluster network provider that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network provider or 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
```

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

```

6.6.5. 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.

6.6.6. 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.

6.6.6.1. 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
...
```

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> 1
```

- 1** 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
```

6.7. SUPPORTED CONFIGURATIONS

The following configurations are supported for the current release of Red Hat OpenShift Service Mesh:

- Red Hat OpenShift Container Platform version 4.x.
- Red Hat Red Hat OpenShift Dedicated version 4.
- Azure Red Hat OpenShift version 4.

**NOTE**

Red Hat OpenShift Online is not supported for Red Hat OpenShift Service Mesh.

- This release of Red Hat OpenShift Service Mesh is only available on OpenShift Container Platform x86_64, IBM Z, and IBM Power Systems.
 - IBM Z is only supported on OpenShift Container Platform 4.6 and later.
 - IBM Power Systems is only supported on OpenShift Container Platform 4.6 and later.
- Configurations where all Service Mesh components are contained within a single OpenShift Container Platform cluster. Red Hat OpenShift Service Mesh does not support management of microservices that reside outside of the cluster within which Service Mesh is running.
- Configurations that do not integrate external services such as virtual machines.

For additional information about Red Hat OpenShift Service Mesh lifecycle and supported configurations, refer to the [Support Policy](#).

6.7.1. Supported network configurations

Red Hat OpenShift Service Mesh supports the following network configurations.

- OpenShift-SDN
- OVN-Kubernetes is supported on OpenShift Container Platform 4.7.32+, OpenShift Container Platform 4.8.12+, and OpenShift Container Platform 4.9+.
- Third-Party Container Network Interface (CNI) plug-ins that have been certified on OpenShift Container Platform and passed Service Mesh conformance testing. See [Certified OpenShift CNI Plug-ins](#) for more information.

6.7.2. Supported configurations for Kiali

- The Kiali observability console is only supported on the two most recent releases of the Chrome, Edge, Firefox, or Safari browsers.

6.7.3. Supported configurations for Distributed Tracing

- Jaeger agent as a sidecar is the only supported configuration for Jaeger. Jaeger as a daemonset is not supported for multitenant installations or OpenShift Dedicated.

6.7.4. Supported Mixer adapters

- This release only supports the following Mixer adapter:
 - 3scale Istio Adapter

6.7.5. Operator overview

Red Hat OpenShift Service Mesh requires the following four Operators:

- **OpenShift Elasticsearch** - (Optional) Provides database storage for tracing and logging with Jaeger. It is based on the open source [Elasticsearch](#) project.

- **Jaeger** - Provides tracing to monitor and troubleshoot transactions in complex distributed systems. It is based on the open source [Jaeger](#) project.
- **Kiali** - Provides observability for your service mesh. Allows you to view configurations, monitor traffic, and analyze traces in a single console. It is based on the open source [Kiali](#) project.
- **Red Hat OpenShift Service Mesh** - Allows you to connect, secure, control, and observe the microservices that comprise your applications. The Service Mesh Operator defines and monitors the **ServiceMeshControlPlane** resources that manage the deployment, updating, and deletion of the Service Mesh components. It is based on the open source [Istio](#) project.

Next steps

- [Install Red Hat OpenShift Service Mesh](#) in your OpenShift Container Platform environment.

6.8. OPTIMIZING ROUTING

The OpenShift Container Platform HAProxy router scales to optimize performance.

6.8.1. Baseline Ingress Controller (router) performance

The OpenShift Container Platform Ingress Controller, or router, is the Ingress point for all external traffic destined for OpenShift Container Platform services.

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

Encryption	LoadBalancerService	HostNetwork
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

Default Ingress Controller configuration with **ROUTER_THREADS=4** was used and two different endpoint publishing strategies (LoadBalancerService/HostNetwork) were tested. TLS session resumption was used for encrypted routes. With HTTP keep-alive, a single HAProxy router is capable of saturating 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 5 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.

6.8.2. Ingress Controller (router) performance optimizations

OpenShift Container Platform no longer supports modifying Ingress Controller deployments by setting environment variables such as **ROUTER_THREADS**, **ROUTER_DEFAULT_TUNNEL_TIMEOUT**, **ROUTER_DEFAULT_CLIENT_TIMEOUT**, **ROUTER_DEFAULT_SERVER_TIMEOUT**, and **RELOAD_INTERVAL**.

You can modify the Ingress Controller deployment, but if the Ingress Operator is enabled, the configuration is overwritten.

6.9. POST-INSTALLATION RHOSP NETWORK CONFIGURATION

You can configure some aspects of an OpenShift Container Platform on Red Hat OpenStack Platform (RHOSP) cluster after installation.

6.9.1. 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> grafana-openshift-monitoring.apps.<cluster name>.<base domain>
<apps_FIP> <app name>.apps.<cluster name>.<base domain>
```

6.9.2. 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 ports to the pool when it is created, such as when a new host is added, or a new namespace is created. 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**.

6.9.3. 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:

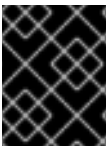
```
$ oc edit networks.operator.openshift.io cluster
```

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 ❶
      poolMinPorts: 1 ❷
      poolBatchPorts: 3 ❸
      poolMaxPorts: 5 ❹
```

- ❶ Set **enablePortPoolsPrepopulation** to **true** to make Kuryr create new Neutron ports after a namespace is created or a new node is added to the cluster. This setting raises the Neutron ports quota but can reduce the time that is required to spawn pods. The default value is **false**.
- ❷ 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**.
- ❸ **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**.
- ❹ 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.

CHAPTER 7. POST-INSTALLATION STORAGE CONFIGURATION

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

7.1. DYNAMIC PROVISIONING

7.1.1. 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 plug-in APIs.

7.1.2. Available dynamic provisioning plug-ins

OpenShift Container Platform provides the following provisioner plug-ins, which have generic implementations for dynamic provisioning that use the cluster's configured provider's API to create new storage resources:

Storage type	Provisioner plug-in 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 plug-in 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 plug-in also requires configuration for the relevant cloud, host, or third-party provider as per the relevant documentation.

7.2. 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 plug-in types.

7.2.1. 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 ❶
apiVersion: storage.k8s.io/v1 ❷
metadata:
  name: <storage-class-name> ❸
  annotations: ❹
    storageclass.kubernetes.io/is-default-class: 'true'
  ...
provisioner: kubernetes.io/aws-ebs ❺
parameters: ❻
  type: gp2
...
```

- ❶ (required) The API object type.
- ❷ (required) The current apiVersion.
- ❸ (required) The name of the storage class.
- ❹ (optional) Annotations for the storage class.
- ❺ (required) The type of provisioner associated with this storage class.
- ❻ (optional) The parameters required for the specific provisioner, this will change from plug-in to plug-in.

7.2.2. 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
...
```

7.2.3. 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-robin 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**.

7.2.4. AWS Elastic Block Store (EBS) object definition

aws-ebs-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
```



```

name: <storage-class-name> ❶
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1 ❷
  iopsPerGB: "10" ❸
  encrypted: "true" ❹
  kmsKeyId: keyvalue ❺
  fsType: ext4 ❻

```

- ❶ (required) Name of the storage class. The persistent volume claim uses this storage class for provisioning the associated persistent volumes.
- ❷ (required) Select from **io1**, **gp2**, **sc1**, **st1**. The default is **gp2**. See the [AWS documentation](#) for valid Amazon Resource Name (ARN) values.
- ❸ Optional: Only for **io1** volumes. I/O operations per second per GiB. The AWS volume plug-in 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.
- ❹ Optional: Denotes whether to encrypt the EBS volume. Valid values are **true** or **false**.
- ❺ 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.
- ❻ 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**.

7.2.5. Azure Disk object definition

azure-advanced-disk-storageclass.yaml

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: <storage-class-name> ❶
provisioner: kubernetes.io/azure-disk
volumeBindingMode: WaitForFirstConsumer ❷
allowVolumeExpansion: true
parameters:
  kind: Managed ❸
  storageaccounttype: Premium_LRS ❹
reclaimPolicy: Delete

```

- ❶ Name of the storage class. The persistent volume claim uses this storage class for provisioning the associated persistent volumes.
- ❷ 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.
- ❸ 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.

7.2.6. 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>
```

Example output

```
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**.

7.2.6.1. 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.

7.2.7. 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**.

7.2.8. VMware vSphere object definition

vsphere-storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
```

```

metadata:
  name: <storage-class-name> ❶
provisioner: kubernetes.io/vsphere-volume ❷
parameters:
  diskformat: thin ❸

```

- ❶ Name of the storage class. The persistent volume claim uses this storage class for provisioning the associated persistent volumes.
- ❷ For more information about using VMware vSphere with OpenShift Container Platform, see the [VMware vSphere documentation](#).
- ❸ **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**.

7.2.9. 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> ❶
  annotations:
    storageclass.kubernetes.io/is-default-class: "<boolean>" ❷
provisioner: csi.ovirt.org
allowVolumeExpansion: <boolean> ❸
reclaimPolicy: Delete ❹
volumeBindingMode: Immediate ❺
parameters:
  storageDomainName: <rhv-storage-domain-name> ❻
  thinProvisioning: "<boolean>" ❼
  csi.storage.k8s.io/fstype: <file_system_type> ❽

```

- ❶ Name of the storage class.
- ❷ 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**.
- ❸ **true** enables dynamic volume expansion, **false** prevents it. **true** is recommended.
- ❹ Dynamically provisioned persistent volumes of this storage class are created with this reclaim policy. This default policy is **Delete**.
- ❺ 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**.

7.3. CHANGING THE DEFAULT STORAGE CLASS

Use the following process to change the default storage class. For example you have two defined storage classes, **gp2** and **standard**, and you want to change the default storage class from **gp2** to **standard**.

1. List the storage class:

```
$ oc get storageclass
```

Example output

NAME	TYPE
gp2 (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 gp2 -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
gp2	kubernetes.io/aws-ebs
standard (default)	kubernetes.io/aws-ebs

7.4. 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.

7.5. AVAILABLE PERSISTENT STORAGE OPTIONS

Understand your persistent storage options so that you can optimize your OpenShift Container Platform environment.

Table 7.1. Available storage options

Storage type	Description	Examples
Block	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> Accessible through a REST API endpoint Configurable for use in the OpenShift Container Platform 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 plug-in.



IMPORTANT

Currently, CNS is not supported in OpenShift Container Platform 4.9.

7.6. RECOMMENDED CONFIGURABLE STORAGE TECHNOLOGY

The following table summarizes the recommended and configurable storage technologies for the given OpenShift Container Platform cluster application.

Table 7.2. Recommended and configurable storage technology

Storage type	ROX ¹	RWX ²	Registry	Scaled registry	Metrics ³	Logging	Apps
Block	Yes ⁴	No	Configurable	Not configurable	Recommended	Recommended	Recommended
File	Yes ⁴	Yes	Configurable	Configurable	Configurable ⁵	Configurable ⁶	Recommended
Object	Yes	Yes	Recommended	Recommended	Not configurable	Not configurable	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, using any shared storage would be an anti-pattern. One volume per elasticsearch is required.

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 Container Platform registry where two or more pod replicas are running.

7.6.1. 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.

7.6.1.1. Registry

In a non-scaled/high-availability (HA) OpenShift Container Platform 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 Container Platform registry cluster deployment with production workloads.

7.6.1.2. Scaled registry

In a scaled/HA OpenShift Container Platform 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.
- 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.

7.6.1.3. 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.

7.6.1.4. Logging

In an OpenShift Container Platform hosted logging cluster deployment:

- The preferred storage technology is block storage.
- Object storage is not configurable.

7.6.1.5. 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.

7.6.2. Other specific application storage recommendations

- OpenShift Container Platform Internal **etcd**: For the best **etcd** reliability, the lowest consistent latency storage technology is preferable.
- It is highly recommended that you use **etcd** with storage that handles serial writes (fsync) quickly, such as NVMe or SSD. Ceph, NFS, and spinning disks are not recommended.
- 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.

7.7. DEPLOY RED HAT OPENSIFT CONTAINER STORAGE

Red Hat OpenShift Container Storage 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 Container Storage is completely integrated with OpenShift Container Platform for deployment, management, and monitoring.

If you are looking for Red Hat OpenShift Container Storage information about...	See the following Red Hat OpenShift Container Storage documentation:
What's new, known issues, notable bug fixes, and Technology Previews	OpenShift Container Storage 4.7 Release Notes
Supported workloads, layouts, hardware and software requirements, sizing and scaling recommendations	Planning your OpenShift Container Storage 4.5 deployment
Instructions on preparing to deploy when your environment is not directly connected to the internet	Preparing to deploy OpenShift Container Storage 4.5 in a disconnected environment

If you are looking for Red Hat OpenShift Container Storage information about...	See the following Red Hat OpenShift Container Storage documentation:
Instructions on deploying OpenShift Container Storage to use an external Red Hat Ceph Storage cluster	Deploying OpenShift Container Storage 4.5 in external mode
Instructions on deploying OpenShift Container Storage to local storage on bare metal infrastructure	Deploying OpenShift Container Storage 4.5 using bare metal infrastructure
Instructions on deploying OpenShift Container Storage on Red Hat OpenShift Container Platform VMWare vSphere clusters	Deploying OpenShift Container Storage 4.5 on VMWare vSphere
Instructions on deploying OpenShift Container Storage using Amazon Web Services for local or cloud storage	Deploying OpenShift Container Storage 4.5 using Amazon Web Services
Instructions on deploying and managing OpenShift Container Storage on existing Red Hat OpenShift Container Platform Google Cloud clusters	Deploying and managing OpenShift Container Storage 4.5 using Google Cloud
Instructions on deploying and managing OpenShift Container Storage on existing Red Hat OpenShift Container Platform Azure clusters	Deploying and managing OpenShift Container Storage 4.5 using Microsoft Azure
Managing a Red Hat OpenShift Container Storage 4.5 cluster	Managing OpenShift Container Storage 4.5
Monitoring a Red Hat OpenShift Container Storage 4.5 cluster	Monitoring Red Hat OpenShift Container Storage 4.5
Resolve issues encountered during operations	Troubleshooting OpenShift Container Storage 4.5
Migrating your OpenShift Container Platform cluster from version 3 to version 4	Migration

CHAPTER 8. 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.

8.1. 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.

8.1.1. 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.

8.1.2. 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](#) .

8.1.3. 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	<p>Defines how new identities are mapped to users when they log in. Enter one of the following values:</p> <p>claim</p> <p>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.</p> <p>lookup</p> <p>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.</p> <p>generate</p> <p>Provisions a user with the identity's preferred user name. If a user with the preferred user name is already mapped to an existing identity, a unique user name is generated. For example, myuser2. This method should not be used in combination with external processes that require exact matches between OpenShift Container Platform user names and identity provider user names, such as LDAP group sync.</p> <p>add</p> <p>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.</p>



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**.

8.1.4. 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:
        fileName:
          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](#).

8.2. USING RBAC TO DEFINE AND APPLY PERMISSIONS

Understand and apply role-based access control.

8.2.1. 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. <i>Cluster roles</i> exist cluster-wide, and <i>cluster role bindings</i> can reference only cluster roles.
Local RBAC	Roles and bindings that are scoped to a given project. While <i>local roles</i> exist only in a single project, local role bindings can reference <i>both</i> 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.

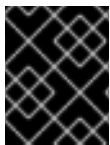
This two-level hierarchy allows reuse across multiple projects through the cluster roles while allowing customization inside of individual projects through local roles.

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.

8.2.1.1. 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

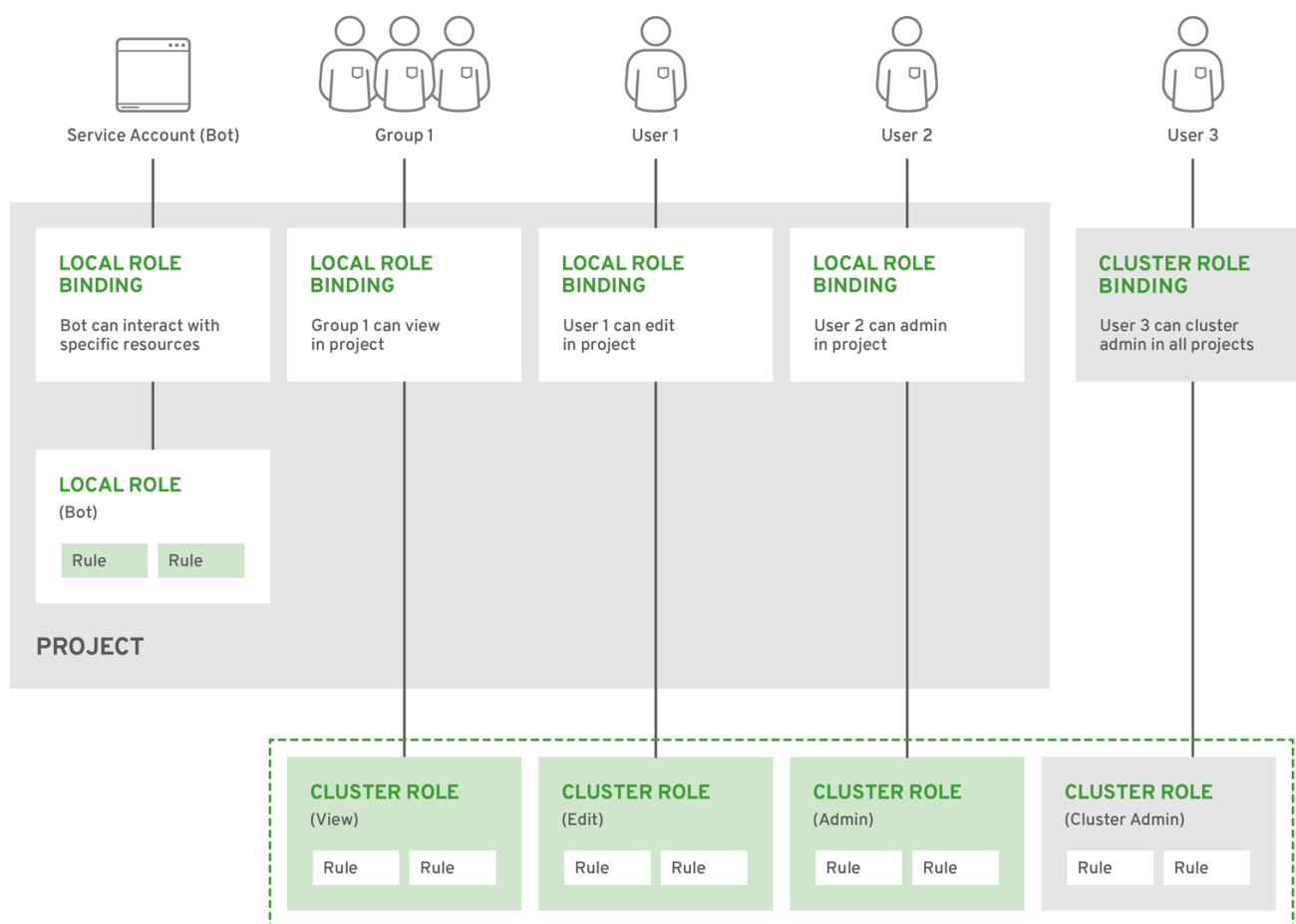
It is not recommended to manually modify the default cluster roles. Modifications to these system roles can prevent a cluster from functioning properly.

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.

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.



OPENSIFT_415489_0218

8.2.1.2. 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.

8.2.1.2.1. 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.

8.2.2. 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.
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.

8.2.3. 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 they 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.

8.2.4. 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

...

```

8.2.5. 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

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	

Group system:serviceaccounts:joe-project

8.2.6. 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**.

8.2.7. 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
```

8.2.8. 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
```

8.2.9. 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 8.1. 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.

8.2.10. 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 8.2. 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.
\$ oc adm policy remove-cluster-role-from-group <role> <groupname>	Removes a given role from specified groups for all projects in the cluster.

8.2.11. 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>
```

8.3. 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-abc.com
INFO Login to the console with user: kubeadmin, password: <provided>
```

8.3.1. 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
```

8.4. IMAGE CONFIGURATION

Understand and configure image registry settings.

8.4.1. 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.

Parameter	Description
allowedRegistriesForImport	<p>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.</p> <p>Every element of this list contains a location of the registry specified by the registry domain name.</p> <p>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.</p> <p>insecure: Insecure indicates whether the registry is secure or insecure. By default, if not otherwise specified, the registry is assumed to be secure.</p>
additionalTrustedCA	<p>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.</p> <p>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.</p>
externalRegistryHostnames	<p>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.</p>

Parameter	Description
registrySources	<p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>containerRuntimeSearchRegistries: Registries for which image pull and push actions are allowed using image short names. All other registries are blocked.</p> <p>Either blockedRegistries or allowedRegistries can be set, but not both.</p>



WARNING

When the **allowedRegistries** parameter is defined, all registries, including **registry.redhat.io** and **quay.io** registries and the default internal 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
-----------	-------------

Parameter	Description
internalRegistryHostname	Set by the Image Registry Operator, which controls the internalRegistryHostname . It sets the hostname for the default internal image registry. The value must be in hostname[:port] format. For backward compatibility, you can still use the OPENSIFT_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.

8.4.2. Configuring image registry settings

You can configure image registry settings by editing the **image.config.openshift.io/cluster** custom resource (CR). The Machine Config Operator (MCO) watches the **image.config.openshift.io/cluster** CR for any changes to the registries and reboots the 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 internal 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
ci-ln-j5cd0qt-f76d1-vfj5x-master-0	Ready	master	98m	v1.22.1
ci-ln-j5cd0qt-f76d1-vfj5x-master-1	Ready,SchedulingDisabled	master	99m	v1.22.1
ci-ln-j5cd0qt-f76d1-vfj5x-master-2	Ready	master	98m	v1.22.1

```

ci-ln-j5cd0qt-f76d1-vfj5x-worker-b-nsnd4 Ready worker 90m v1.22.1
ci-ln-j5cd0qt-f76d1-vfj5x-worker-c-5z2gz NotReady,SchedulingDisabled worker 90m
v1.22.1
ci-ln-j5cd0qt-f76d1-vfj5x-worker-d-stsjv Ready worker 90m v1.22.1

```

For more information on the allowed, blocked, and insecure registry parameters, see [Configuring image registry settings](#).

8.4.2.1. 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 base64-encoded certificate 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
```

8.4.2.2. 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 restricted networks 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 restricted network.
- 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 post-installation 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
    source: registry.access.redhat.com/ubi8/ubi-minimal 2
  - mirrors:
    - example.com/example/ubi-minimal
    source: registry.access.redhat.com/ubi8/ubi-minimal
  - mirrors:
    - mirror.example.com/redhat
    source: registry.redhat.io/openshift4 3
```

1 Indicates the name of the image registry and repository.

2 Indicates the registry and repository containing the content that is mirrored.

3

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

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	v1.22.1
ip-10-0-138-148.ec2.internal	Ready	master	11m	v1.22.1
ip-10-0-139-122.ec2.internal	Ready	master	11m	v1.22.1
ip-10-0-147-35.ec2.internal	Ready,SchedulingDisabled	worker	7m	v1.22.1
ip-10-0-153-12.ec2.internal	Ready	worker	7m	v1.22.1
ip-10-0-154-10.ec2.internal	Ready	master	11m	v1.22.1

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

- 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. Access the node's files:

```
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"]
[[registry]]
  location = "registry.access.redhat.com/ubi8/"
  insecure = false
```



```

blocked = false
mirror-by-digest-only = true
prefix = ""

[[registry.mirror]]
location = "example.io/example/ubi8-minimal"
insecure = false

[[registry.mirror]]
location = "example.com/example/ubi8-minimal"
insecure = false

```

- 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.

8.5. 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.

8.5.1. Prerequisites

- [Mirroring Operator catalogs for use with disconnected clusters](#)

8.5.2. 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.

8.5.3. 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.

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 <.>
  namespace: openshift-marketplace <.>
spec:
  sourceType: grpc
  image: <registry>/<namespace>/redhat-operator-index:v4.9 <.>
  displayName: My Operator Catalog
  publisher: <publisher_name> <.>
  updateStrategy:
    registryPoll: <.>
    interval: 30m
```

<.> 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. <.> 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. <.> Specify your index image. <.> Specify your name or an organization name publishing the catalog. <.> 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.

- If your index image is hosted on a private registry and requires authentication, see [Accessing images for Operators from private registries](#).
- If you want your catalogs to be able to automatically update their index image version after cluster upgrades by using Kubernetes version-based image tags, see [Image template for custom catalog sources](#).

8.6. 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 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.

8.6.1. Installing from OperatorHub using the web console

You can install and subscribe to an Operator from OperatorHub 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.

8.6.2. Installing from OperatorHub using the CLI

Instead of using the OpenShift Container Platform web console, you can install an Operator from OperatorHub 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**, then the **openshift-operators** namespace already has an appropriate 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.

- 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>

```

- 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

```

- 1** For **AllNamespaces** install mode usage, specify the **openshift-operators** namespace. 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.

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 9. 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.

9.1. SENDING NOTIFICATIONS TO EXTERNAL SYSTEMS

In OpenShift Container Platform 4.9, 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.

9.1.1. 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** role.

Procedure

1. In the **Administrator** perspective, navigate to **Administration** → **Cluster Settings** → **Configuration** → **Alertmanager**.



NOTE

Alternatively, you can navigate 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. Select **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. Select **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. Select **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. Choose whether TLS is required.
 - d. Select **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 will be sent to the receiver. If you want label values for firing alerts to be matched exactly before they are sent to the receiver:
 - 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. Select **Add Label** to add further routing labels.
6. Select **Create** to create the receiver.

9.2. ADDITIONAL RESOURCES

- [Understanding the monitoring stack](#)
- [Managing alerts](#)

CHAPTER 10. 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.



IMPORTANT

The process for converting a cluster from connected to disconnected 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 <https://access.redhat.com/support/offerings/techpreview/>.

10.1. 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 internal registry of the OpenShift Container Platform cluster 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.

10.2. 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.

10.3. 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":"b3BlbnNoaWZ0Y3UjhGOVZPT0IOMEFaUjdPUzRGTA==","email":"user@exa
mple.com"},
"quay.io":
{"auth":"b3BlbnNoaWZ0LXJlbGVhc2UzZGOVZPT0IOMEFaUGSTd4VGVGUVUjdPUzR
GTA==","email":"user@example.com"},
"registry.connect.redhat.com"
{"auth":"NTE3MTMwNDB8dWhjLTFEZIN3VHkxOSTd4VGVGUVU1MdTpleUpoYkdjaUail
A==","email":"user@example.com"},
"registry.redhat.io":
{"auth":"NTE3MTMwNDB8dWhjLTFEZIN3VH3BGSTd4VGVGUVU1MdTpleUpoYkdjaU9
fZw==","email":"user@example.com"},
"registry.svc.ci.openshift.org":
{"auth":"dXNlcjpyWjAwVWFjSEJiT2RKVW1pSmg4dW92dGp1SXRxQ3RGN1pwajJhN1
ZXeTRV"},
"my-registry:5000/my-namespace/{
"auth":"dXNlcjpyWjAwVWFjSEJiT2RKVW1pSmg4dW92dGp1SXRxQ3RGN1pwajJhN1
ZXeTRV"}}
```

10.4. 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 you want to mirror. For example, this might be a pruned index image that you created previously, or one of the source index images for the default catalogs, such as **{index-image-pullspec}**.

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](#).

10.5. 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:

```
$ 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 1
    source: quay.io/openshift-release-dev/ocp-release 2
  - mirrors:
    - mirror.registry.com:443/ocp/release
    source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

1 Specifies the name of the mirror image registry and repository.

2 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="}}}
```

- 1 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
```

- 1 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
```

10.6. 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	STATUS	RESTARTS	AGE	NAME	READY
kube-system	1/1 Running	0	39m	apiserver-watcher-ci-ln-47ltxb-f76d1-mrffg-master-0	
kube-system	1/1 Running	0	39m	apiserver-watcher-ci-ln-47ltxb-f76d1-mrffg-master-1	
kube-system	1/1 Running	0	39m	apiserver-watcher-ci-ln-47ltxb-f76d1-mrffg-master-2	
openshift-apiserver-operator	1/1 Running	3	45m	openshift-apiserver-operator-79c7c646fd-5rvr5	
openshift-apiserver	Running	0	29m	apiserver-b944c4645-q694g	2/2
openshift-apiserver	Running	0	31m	apiserver-b944c4645-shdxb	2/2
openshift-apiserver	Running	0	33m	apiserver-b944c4645-x7rf2	2/2
...					

- Ensure your nodes are in the READY status:

```
$ oc get nodes
```

Example output

NAME	STATUS	ROLES	AGE	VERSION
ci-ln-47ltxb-f76d1-mrffg-master-0	Ready	master	42m	v1.21.1+a620f50
ci-ln-47ltxb-f76d1-mrffg-master-1	Ready	master	42m	v1.21.1+a620f50
ci-ln-47ltxb-f76d1-mrffg-master-2	Ready	master	42m	v1.21.1+a620f50
ci-ln-47ltxb-f76d1-mrffg-worker-a-gsxbz	Ready	worker	35m	v1.21.1+a620f50
ci-ln-47ltxb-f76d1-mrffg-worker-b-5qqdx	Ready	worker	35m	v1.21.1+a620f50
ci-ln-47ltxb-f76d1-mrffg-worker-c-rjqpq	Ready	worker	34m	v1.21.1+a620f50

10.7. 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.

10.8. 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

10.9. 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"] 1
```

- 1** The **registry** and **registry.mirror** entries created by the ICSPs you deleted are removed.