



OpenShift Container Platform 4.20

Postinstallation configuration

Day 2 operations for OpenShift Container Platform

OpenShift Container Platform 4.20 Postinstallation configuration

Day 2 operations for OpenShift Container Platform

Legal Notice

Copyright © Red Hat.

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, 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 Software Collections 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. POSTINSTALLATION CONFIGURATION OVERVIEW	8
1.1. POSTINSTALLATION CONFIGURATION TASKS	8
CHAPTER 2. CONFIGURING A PRIVATE CLUSTER	10
2.1. ABOUT PRIVATE CLUSTERS	10
2.1.1. DNS	10
2.1.2. Ingress Controller	10
2.1.3. API server	10
2.2. CONFIGURING DNS RECORDS TO BE PUBLISHED IN A PRIVATE ZONE	11
2.3. SETTING THE INGRESS CONTROLLER TO PRIVATE	12
2.4. RESTRICTING THE API SERVER TO PRIVATE	13
2.5. CONFIGURING A PRIVATE STORAGE ENDPOINT ON AZURE	15
2.5.1. Limitations for configuring a private storage endpoint on Azure	15
2.5.2. Configuring a private storage endpoint on Azure by enabling the Image Registry Operator to discover VNet and subnet names	16
2.5.3. Configuring a private storage endpoint on Azure with user-provided VNet and subnet names	17
2.5.4. Optional: Disabling redirect when using a private storage endpoint on Azure	19
CHAPTER 3. CONFIGURING MULTI-ARCHITECTURE COMPUTE MACHINES ON AN OPENSIFT CLUSTER .	21
3.1. ABOUT CLUSTERS WITH MULTI-ARCHITECTURE COMPUTE MACHINES	21
3.1.1. Configuring your cluster with multi-architecture compute machines	21
3.2. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINE ON AZURE	23
3.2.1. Verifying cluster compatibility	23
3.2.2. Creating a 64-bit ARM boot image using the Azure image gallery	24
3.2.3. Creating a 64-bit x86 boot image using the Azure image gallery	26
3.2.4. Adding a multi-architecture compute machine set to your Azure cluster	28
3.3. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES ON AWS	31
3.3.1. Verifying cluster compatibility	31
3.3.2. Adding a multi-architecture compute machine set to your AWS cluster	32
3.4. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES ON GOOGLE CLOUD	35
3.4.1. Verifying cluster compatibility	35
3.4.2. Adding a multi-architecture compute machine set to your Google Cloud cluster	36
3.5. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES ON BARE METAL, IBM POWER, OR IBM Z	39
3.5.1. Verifying cluster compatibility	40
3.5.2. Creating RHCOS machines using an ISO image	41
3.5.3. Creating RHCOS machines by PXE or iPXE booting	42
3.5.4. Approving the certificate signing requests for your machines	45
3.6. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES ON IBM Z AND IBM LINUXONE WITH Z/VM	47
3.6.1. Verifying cluster compatibility	48
3.6.2. Creating RHCOS machines on IBM Z with z/VM	49
3.6.3. Approving the certificate signing requests for your machines	52
3.7. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES ON IBM Z AND IBM LINUXONE IN AN LPAR	54
3.7.1. Verifying cluster compatibility	55
3.7.2. Creating RHCOS machines on IBM Z in an LPAR	56
3.7.3. Approving the certificate signing requests for your machines	58
3.8. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES ON IBM Z AND IBM LINUXONE WITH RHEL KVM	61
3.8.1. Verifying cluster compatibility	62

3.8.2. Creating RHCOS machines using virt-install	63
3.8.3. Approving the certificate signing requests for your machines	64
3.9. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES ON IBM POWER	67
3.9.1. Verifying cluster compatibility	68
3.9.2. Creating RHCOS machines using an ISO image	69
3.9.3. Creating RHCOS machines by PXE or iPXE booting	70
3.9.4. Approving the certificate signing requests for your machines	73
3.10. MANAGING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES	76
3.10.1. Scheduling workloads on clusters with multi-architecture compute machines	76
3.10.1.1. Sample multi-architecture node workload deployments	76
3.10.2. Enabling 64k pages on the Red Hat Enterprise Linux CoreOS (RHCOS) kernel	79
3.10.3. Importing manifest lists in image streams on your multi-architecture compute machines	81
3.11. MANAGING WORKLOADS ON MULTI-ARCHITECTURE CLUSTERS BY USING THE MULTIARCH TUNING OPERATOR	82
3.11.1. Installing the Multiarch Tuning Operator by using the CLI	83
3.11.2. Installing the Multiarch Tuning Operator by using the web console	85
3.11.3. Multiarch Tuning Operator pod labels and architecture support overview	86
3.11.4. Creating the ClusterPodPlacementConfig object	87
3.11.4.1. Creating the ClusterPodPlacementConfig object by using the CLI	89
3.11.4.2. Creating the ClusterPodPlacementConfig object by using the web console	90
3.11.5. Deleting the ClusterPodPlacementConfig object by using the CLI	91
3.11.6. Deleting the ClusterPodPlacementConfig object by using the web console	92
3.11.7. Uninstalling the Multiarch Tuning Operator by using the CLI	92
3.11.8. Uninstalling the Multiarch Tuning Operator by using the web console	94
3.12. MULTIARCH TUNING OPERATOR RELEASE NOTES	95
3.12.1. Release notes for the Multiarch Tuning Operator 1.2.1	95
3.12.1.1. Bug fixes	95
3.12.2. Release notes for the Multiarch Tuning Operator 1.2.0	95
3.12.2.1. New features and enhancements	95
3.12.2.2. Bug fixes	95
3.12.3. Release notes for the Multiarch Tuning Operator 1.1.1	96
3.12.3.1. Bug fixes	96
3.12.4. Release notes for the Multiarch Tuning Operator 1.1.0	96
3.12.4.1. New features and enhancements	96
3.12.4.1.1. Bug fixes	97
3.12.5. Release notes for the Multiarch Tuning Operator 1.0.0	97
3.12.5.1. New features and enhancements	97
CHAPTER 4. POSTINSTALLATION CLUSTER TASKS	98
4.1. AVAILABLE CLUSTER CUSTOMIZATIONS	98
4.1.1. Cluster configuration resources	98
4.1.2. Operator configuration resources	99
4.1.3. Additional configuration resources	99
4.1.4. Informational Resources	100
4.2. ADDING WORKER NODES	100
4.2.1. Adding worker nodes to an on-premise cluster	100
4.2.2. Adding worker nodes to installer-provisioned infrastructure clusters	101
4.2.3. Adding worker nodes to user-provisioned infrastructure clusters	101
4.2.4. Adding worker nodes to clusters managed by the Assisted Installer	101
4.2.5. Adding worker nodes to clusters managed by the multicluster engine for Kubernetes	101
4.3. ADJUST WORKER NODES	101
4.3.1. Understanding the difference between compute machine sets and the machine config pool	102
4.3.2. Scaling a compute machine set manually	102

4.3.3. The compute machine set deletion policy	103
4.3.4. Creating default cluster-wide node selectors	104
4.4. IMPROVING CLUSTER STABILITY IN HIGH LATENCY ENVIRONMENTS USING WORKER LATENCY PROFILES	107
4.4.1. Understanding worker latency profiles	108
4.4.2. Using and changing worker latency profiles	110
4.5. MANAGING CONTROL PLANE MACHINES	113
4.5.1. Adding a control plane node to your cluster	113
4.6. CREATING INFRASTRUCTURE MACHINE SETS FOR PRODUCTION ENVIRONMENTS	119
4.6.1. Creating a compute machine set	119
4.6.2. Creating an infrastructure node	121
4.6.3. Creating a machine config pool for infrastructure machines	123
4.7. ASSIGNING MACHINE SET RESOURCES TO INFRASTRUCTURE NODES	126
4.7.1. Binding infrastructure node workloads using taints and tolerations	126
4.8. MOVING RESOURCES TO INFRASTRUCTURE MACHINE SETS	129
4.8.1. Moving the router	129
4.8.2. Moving the default registry	131
4.8.3. Moving the monitoring solution	132
4.9. ABOUT THE CLUSTER AUTOSCALER	134
4.9.1. Automatic node removal	135
4.9.2. Limitations	136
4.9.3. Interaction with other scheduling features	136
4.9.4. Cluster autoscaler resource definition	136
4.9.5. Deploying a cluster autoscaler	140
4.10. APPLYING AUTOSCALING TO YOUR CLUSTER	140
4.11. ENABLING TECHNOLOGY PREVIEW FEATURES USING FEATUREGATES	140
4.11.1. Understanding feature gates	140
4.11.2. Enabling feature sets using the web console	144
4.11.3. Enabling feature sets using the CLI	146
4.12. ETCD TASKS	147
4.12.1. About etcd encryption	147
4.12.2. Supported encryption types	148
4.12.3. Enabling etcd encryption	148
4.12.4. Disabling etcd encryption	150
4.12.5. Backing up etcd data	151
4.12.6. Defragmenting etcd data	153
4.12.6.1. Automatic defragmentation	154
4.12.6.2. Manual defragmentation	154
4.12.7. Restoring to a previous cluster state for more than one node	157
4.12.8. Issues and workarounds for restoring a persistent storage state	159
4.13. POD DISRUPTION BUDGETS	160
4.13.1. Understanding how to use pod disruption budgets to specify the number of pods that must be up	160
4.13.2. Specifying the number of pods that must be up with pod disruption budgets	162
4.13.3. Specifying the eviction policy for unhealthy pods	163
CHAPTER 5. POSTINSTALLATION NODE TASKS	165
5.1. ADDING RHCOS COMPUTE MACHINES TO AN OPENSIFT CONTAINER PLATFORM CLUSTER	165
5.1.1. Prerequisites	165
5.1.2. Creating RHCOS machines using an ISO image	165
5.1.3. Creating RHCOS machines by PXE or iPXE booting	167
5.1.4. Approving the certificate signing requests for your machines	169
5.1.5. Adding a new RHCOS worker node with a custom /var partition in AWS	172
5.2. DEPLOYING MACHINE HEALTH CHECKS	177

5.2.1. About machine health checks	177
5.2.1.1. Limitations when deploying machine health checks	178
5.2.2. Sample MachineHealthCheck resource	178
5.2.2.1. Short-circuiting machine health check remediation	179
5.2.2.1.1. Setting maxUnhealthy by using an absolute value	180
5.2.2.1.2. Setting maxUnhealthy by using percentages	180
5.2.3. Creating a machine health check resource	180
5.2.4. Scaling a compute machine set manually	181
5.2.5. Understanding the difference between compute machine sets and the machine config pool	182
5.3. RECOMMENDED NODE HOST PRACTICES	182
5.3.1. Creating a KubeletConfig CR to edit kubelet parameters	183
5.3.2. Modifying the number of unavailable worker nodes	188
5.3.3. Control plane node sizing	189
5.3.4. Setting up CPU Manager	191
5.4. HUGE PAGES	196
5.4.1. What huge pages do	196
5.4.2. How huge pages are consumed by apps	196
5.4.3. Configuring huge pages at boot time	197
5.5. UNDERSTANDING DEVICE PLUGINS	199
5.5.1. Example device plugins	200
5.5.2. Methods for deploying a device plugin	200
5.5.3. Understanding the Device Manager	200
5.5.4. Enabling Device Manager	201
5.6. TAINTS AND TOLERATIONS	202
5.6.1. Understanding taints and tolerations	202
5.6.2. Adding taints and tolerations	205
5.6.3. Adding taints and tolerations using a compute machine set	207
5.6.4. Binding a user to a node using taints and tolerations	209
5.6.5. Controlling nodes with special hardware using taints and tolerations	210
5.6.6. Removing taints and tolerations	211
5.7. TOPOLOGY MANAGER	212
5.7.1. Topology Manager policies	212
5.7.2. Setting up Topology Manager	212
5.7.3. Pod interactions with Topology Manager policies	213
5.8. RESOURCE REQUESTS AND OVERCOMMITMENT	214
5.9. CLUSTER-LEVEL OVERCOMMIT USING THE CLUSTER RESOURCE OVERRIDE OPERATOR	214
5.9.1. Installing the Cluster Resource Override Operator using the web console	217
5.9.2. Installing the Cluster Resource Override Operator using the CLI	219
5.9.3. Configuring cluster-level overcommit	222
5.10. NODE-LEVEL OVERCOMMIT	224
5.10.1. Understanding container CPU and memory requests	224
5.10.2. Understanding overcommitment and quality of service classes	224
5.10.2.1. Understanding how to reserve memory across quality of service tiers	225
5.10.3. Understanding swap memory and QoS	225
5.10.4. Understanding nodes overcommitment	226
5.10.5. Disabling or enforcing CPU limits using CPU CFS quotas	227
5.10.6. Reserving resources for system processes	228
5.10.7. Disabling overcommitment for a node	228
5.11. PROJECT-LEVEL LIMITS	228
5.11.1. Disabling overcommitment for a project	229
5.12. FREEING NODE RESOURCES USING GARBAGE COLLECTION	229
5.12.1. Understanding how terminated containers are removed through garbage collection	229
5.12.2. Understanding how images are removed through garbage collection	230

5.12.3. Configuring garbage collection for containers and images	231
5.13. USING THE NODE TUNING OPERATOR	234
5.13.1. Accessing an example Node Tuning Operator specification	235
5.13.2. Custom tuning specification	235
5.13.3. Default profiles set on a cluster	240
5.13.4. Supported TuneD daemon plugins	241
5.14. CONFIGURING THE MAXIMUM NUMBER OF PODS PER NODE	242
5.15. MACHINE SCALING WITH STATIC IP ADDRESSES	244
5.15.1. Scaling machines to use static IP addresses	244
5.15.2. Machine set scaling of machines with configured static IP addresses	245
5.15.3. Using a machine set to scale machines with configured static IP addresses	246
CHAPTER 6. POSTINSTALLATION NETWORK CONFIGURATION	250
6.1. USING THE CLUSTER NETWORK OPERATOR	250
6.2. NETWORK CONFIGURATION TASKS	250
6.2.1. Creating default network policies for a new project	250
6.2.1.1. Modifying the template for new projects	250
6.2.1.2. Adding network policies to the new project template	251
CHAPTER 7. CONFIGURING IMAGE STREAMS AND IMAGE REGISTRIES	254
7.1. CONFIGURING IMAGE STREAMS FOR A DISCONNECTED CLUSTER	254
7.1.1. Cluster Samples Operator assistance for mirroring	254
7.1.2. Using Cluster Samples Operator image streams with alternate or mirrored registries	255
7.1.3. Preparing your cluster to gather support data	256
7.2. CONFIGURING PERIODIC IMPORTING OF CLUSTER SAMPLE OPERATOR IMAGE STREAM TAGS	257
CHAPTER 8. POSTINSTALLATION STORAGE CONFIGURATION	259
8.1. DYNAMIC PROVISIONING	259
8.2. RECOMMENDED CONFIGURABLE STORAGE TECHNOLOGY	259
8.2.1. Specific application storage recommendations	260
8.2.1.1. Registry	261
8.2.1.2. Scaled registry	261
8.2.1.3. Metrics	261
8.2.1.4. Logging	262
8.2.1.5. Applications	262
8.2.2. Other specific application storage recommendations	262
8.3. DEPLOY RED HAT OPENSIFT DATA FOUNDATION	263
CHAPTER 9. PREPARING FOR USERS	265
9.1. UNDERSTANDING IDENTITY PROVIDER CONFIGURATION	265
9.1.1. About identity providers in OpenShift Container Platform	265
9.1.2. Supported identity providers	265
9.1.3. Identity provider parameters	266
9.1.4. Sample identity provider CR	266
9.2. USING RBAC TO DEFINE AND APPLY PERMISSIONS	267
9.2.1. RBAC overview	267
9.2.1.1. Default cluster roles	268
9.2.1.2. Evaluating authorization	270
9.2.1.2.1. Cluster role aggregation	271
9.2.2. Projects and namespaces	271
9.2.3. Default projects	272
9.2.4. Viewing cluster roles and bindings	272
9.2.5. Viewing local roles and bindings	279
9.2.6. Adding roles to users	280

9.2.7. Creating a local role	283
9.2.8. Creating a cluster role	283
9.2.9. Local role binding commands	284
9.2.10. Cluster role binding commands	284
9.2.11. Creating a cluster admin	285
9.2.12. Cluster role bindings for unauthenticated groups	285
9.2.13. Adding unauthenticated groups to cluster roles	285
9.3. THE KUBEADMIN USER	286
9.3.1. Removing the kubeadmin user	287
9.4. POPULATING THE SOFTWARE CATALOG FROM MIRRORRED OPERATOR CATALOGS	287
9.4.1. Prerequisites	287
9.4.1.1. Creating the ImageContentSourcePolicy object	288
9.4.1.2. Adding a catalog source to a cluster	288
9.5. ABOUT OPERATOR INSTALLATION FROM THE SOFTWARE CATALOG	290
9.5.1. Installing from the software catalog by using the web console	291
9.5.2. Installing from the software catalog by using the CLI	293
CHAPTER 10. CHANGING THE CLOUD PROVIDER CREDENTIALS CONFIGURATION	301
10.1. ROTATING CLOUD PROVIDER SERVICE KEYS WITH THE CLOUD CREDENTIAL OPERATOR UTILITY	301
10.1.1. Rotating AWS OIDC bound service account signer keys	301
10.1.2. Rotating Google Cloud OIDC bound service account signer keys	305
10.1.3. Rotating Azure OIDC bound service account signer keys	309
10.1.4. Rotating IBM Cloud credentials	313
10.2. ROTATING CLOUD PROVIDER CREDENTIALS	313
10.2.1. Rotating cloud provider credentials manually	313
10.3. REMOVING CLOUD PROVIDER CREDENTIALS	316
10.3.1. Removing cloud provider credentials	316
10.4. ENABLING TOKEN-BASED AUTHENTICATION	317
10.4.1. Configuring the Cloud Credential Operator utility	317
10.4.2. Enabling Microsoft Entra Workload ID on an existing cluster	319
10.4.3. Enabling AWS Security Token Service (STS) on an existing cluster	323
10.4.4. Verifying that a cluster uses short-term credentials	327
10.5. ADDITIONAL RESOURCES	328
CHAPTER 11. CONFIGURING ALERT NOTIFICATIONS	329
11.1. SENDING NOTIFICATIONS TO EXTERNAL SYSTEMS	329
11.2. ADDITIONAL RESOURCES	329
CHAPTER 12. CONVERTING A CONNECTED CLUSTER TO A DISCONNECTED CLUSTER	330

CHAPTER 1. POSTINSTALLATION CONFIGURATION OVERVIEW

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

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

1.1. POSTINSTALLATION CONFIGURATION TASKS

You can perform the postinstallation configuration tasks to configure your environment to meet your needs.

The following lists details these configurations:

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

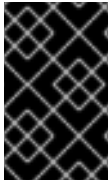
- Ingress Controller
- API server
- [Perform node operations](#): By default, OpenShift Container Platform uses Red Hat Enterprise Linux CoreOS (RHCOS) compute machines. You can perform the following node operations:
 - Add and remove compute machines.
 - Add and remove taints and tolerations.
 - Configure the maximum number of pods per node.
 - Enable Device Manager.
- [Configure users](#): Users can authenticate themselves to the API by using OAuth access tokens. You can configure OAuth to perform the following tasks:
 - Specify an identity provider.
 - Use role-based access control to define and grant permissions to users.
 - Install an Operator from the software catalog.
- [Configuring alert notifications](#): By default, firing alerts are displayed on the Alerting UI of the web console. You can also configure OpenShift Container Platform to send alert notifications to external systems.

CHAPTER 2. CONFIGURING A PRIVATE CLUSTER

After you install an OpenShift Container Platform version 4.20 cluster, you can set some of its core components to be private.

2.1. ABOUT PRIVATE CLUSTERS

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



IMPORTANT

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

2.1.1. 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.

2.1.2. Ingress Controller

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

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

2.1.3. 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, 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. CONFIGURING DNS RECORDS TO BE PUBLISHED IN A PRIVATE ZONE

For all OpenShift Container Platform clusters, whether public or private, DNS records are published in a public zone by default.

You can remove the public zone from the cluster DNS configuration to avoid exposing DNS records to the public. You might want to avoid exposing sensitive information, such as internal domain names, internal IP addresses, or the number of clusters at an organization, or you might simply have no need to publish records publicly. If all the clients that should be able to connect to services within the cluster use a private DNS service that has the DNS records from the private zone, then there is no need to have a public DNS record for the cluster.

After you deploy a cluster, you can modify its DNS to use only a private zone by modifying the **DNS** custom resource (CR). Modifying the **DNS** CR in this way means that any DNS records that are subsequently created are not published to public DNS servers, which keeps knowledge of the DNS records isolated to internal users. This can be done when you configure the cluster to be private, or if you never want DNS records to be publicly resolvable.

Alternatively, even in a private cluster, you might keep the public zone for DNS records because it allows clients to resolve DNS names for applications running on that cluster. For example, an organization can have machines that connect to the public internet and then establish VPN connections for certain private IP ranges in order to connect to private IP addresses. The DNS lookups from these machines use the public DNS to determine the private addresses of those services, and then connect to the private addresses over the VPN.

Procedure

1. Review the **DNS** CR for your cluster by running the following command and observing the output:

```
$ 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** CR to remove the public zone by running the following command:

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

Example output

```
dns.config.openshift.io/cluster patched
```

The Ingress Operator consults the **DNS** CR definition when it creates DNS records for **IngressController** objects. If only private zones are specified, only private records are created.



IMPORTANT

Existing DNS records are not modified when you remove the public zone. You must manually delete previously published public DNS records if you no longer want them to be published publicly.

Verification

- Review the **DNS** CR for your cluster and confirm that the public zone was removed, by running the following command and observing the output:

```
$ 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

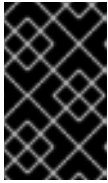
- Install the OpenShift CLI (**oc**).
- Have access to the web console as a user with **admin** privileges.

Procedure

1. In the web portal or console for your cloud provider, take the following actions:
 - a. Locate and delete the appropriate load balancer component:
 - AWS clusters: 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.
 - Azure: Delete the following resources:
 - The **api-v4** rule for the public load balancer.
 - The **frontendIPConfiguration** parameter that is associated with the **api-v4** rule for the public load balancer.
 - The public IP that is specified in the **frontendIPConfiguration** parameter.
 - b. Azure clusters: Configure the Ingress Controller endpoint publishing scope to **Internal**. For more information, see "Configuring the Ingress Controller endpoint publishing scope to Internal".

- c. Delete the **api.\$clustername.\$yourdomain** or **api.\$clustername** DNS entry in the public zone.

2. AWS clusters: Remove the external load balancers:



IMPORTANT

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

- If your cluster uses a control plane machine set, delete the lines in the control plane machine set custom resource that configure your public or external load balancer:

```
# ...
providerSpec:
  value:
    # ...
    loadBalancers:
      - name: lk4pj-ext 1
        type: network 2
      - name: lk4pj-int
        type: network
    # ...
```

- 1 Delete the **name** value for the external load balancer, which ends in **-ext**.
- 2 Delete the **type** value for the external load balancer.

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

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

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

Example output

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

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

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

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

```
$ oc edit machines -n openshift-machine-api <control_plane_name> 1
```

- 1 Specify the name of the control plane machine object to modify.
- B. Remove the lines that describe the external load balancer, which are marked in the following example:

```
# ...
providerSpec:
  value:
# ...
  loadBalancers:
    - name: lk4pj-ext 1
      type: network 2
    - name: lk4pj-int
      type: network
# ...
```

- 1 Delete the **name** value for the external load balancer, which ends in **-ext**.
 - 2 Delete the **type** value for the external load balancer.
- C. Save your changes and exit the object specification.
 - D. Repeat this process for each of the control plane machines.

Additional resources

- [Configuring the Ingress Controller endpoint publishing scope to Internal](#)

2.5. CONFIGURING A PRIVATE STORAGE ENDPOINT ON AZURE

You can leverage the Image Registry Operator to use private endpoints on Azure, which enables seamless configuration of private storage accounts when OpenShift Container Platform is deployed on private Azure clusters. This allows you to deploy the image registry without exposing public-facing storage endpoints.



IMPORTANT

Do not configure a private storage endpoint on Microsoft Azure Red Hat OpenShift (ARO), because the endpoint can put your Microsoft Azure Red Hat OpenShift cluster in an unrecoverable state.

You can configure the Image Registry Operator to use private storage endpoints on Azure in one of two ways:

- By configuring the Image Registry Operator to discover the VNet and subnet names
- With user-provided Azure Virtual Network (VNet) and subnet names

2.5.1. Limitations for configuring a private storage endpoint on Azure

The following limitations apply when configuring a private storage endpoint on Azure:

- When configuring the Image Registry Operator to use a private storage endpoint, public network access to the storage account is disabled. Consequently, pulling images from the registry outside of OpenShift Container Platform only works by setting **disableRedirect: true** in the registry Operator configuration. With redirect enabled, the registry redirects the client to pull images directly from the storage account, which will no longer work due to disabled public network access. For more information, see "Disabling redirect when using a private storage endpoint on Azure".
- This operation cannot be undone by the Image Registry Operator.

2.5.2. Configuring a private storage endpoint on Azure by enabling the Image Registry Operator to discover VNet and subnet names

The following procedure shows you how to set up a private storage endpoint on Azure by configuring the Image Registry Operator to discover VNet and subnet names.

Prerequisites

- You have configured the image registry to run on Azure.
- Your network has been set up using the Installer Provisioned Infrastructure installation method. For users with a custom network setup, see "Configuring a private storage endpoint on Azure with user-provided VNet and subnet names".

Procedure

1. Edit the Image Registry Operator **config** object and set **networkAccess.type** to **Internal**:

```
$ oc edit configs.imageregistry/cluster
```

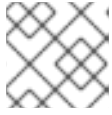
```
# ...
spec:
  # ...
  storage:
    azure:
      # ...
      networkAccess:
        type: Internal
# ...
```

2. Optional: Enter the following command to confirm that the Operator has completed provisioning. This might take a few minutes.

```
$ oc get configs.imageregistry/cluster -o=jsonpath="{.spec.storage.azure.privateEndpointName}" -w
```

3. Optional: If the registry is exposed by a route, and you are configuring your storage account to be private, you must disable redirect if you want pulls external to the cluster to continue to work. Enter the following command to disable redirect on the Image Operator configuration:

```
$ oc patch configs.imageregistry cluster --type=merge -p '{"spec":{"disableRedirect": true}}'
```

**NOTE**

When redirect is enabled, pulling images from outside of the cluster will not work.

Verification

1. Fetch the registry service name by running the following command:

```
$ oc get imagestream -n openshift
```

Example output

```
NAME IMAGE REPOSITORY TAGS UPDATED
cli image-registry.openshift-image-registry.svc:5000/openshift/cli latest 8 hours ago
...
```

2. Enter debug mode by running the following command:

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

3. Run the suggested **chroot** command. For example:

```
$ chroot /host
```

4. Enter the following command to log in to your container registry:

```
$ podman login --tls-verify=false -u unused -p $(oc whoami -t) image-registry.openshift-
image-registry.svc:5000
```

Example output

```
Login Succeeded!
```

5. Enter the following command to verify that you can pull an image from the registry:

```
$ podman pull --tls-verify=false image-registry.openshift-image-
registry.svc:5000/openshift/tools
```

Example output

```
Trying to pull image-registry.openshift-image-
registry.svc:5000/openshift/tools/openshift/tools...
Getting image source signatures
Copying blob 6b245f040973 done
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
22667f53682a2920948d19c7133ab1c9c3f745805c14125859d20cede07f11f9
```

2.5.3. Configuring a private storage endpoint on Azure with user-provided VNet and subnet names

Use the following procedure to configure a storage account that has public network access disabled and is exposed behind a private storage endpoint on Azure.

Prerequisites

- You have configured the image registry to run on Azure.
- You must know the VNet and subnet names used for your Azure environment.
- If your network was configured in a separate resource group in Azure, you must also know its name.

Procedure

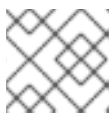
1. Edit the Image Registry Operator **config** object and configure the private endpoint using your VNet and subnet names:

```
$ oc edit configs.imageregistry/cluster

# ...
spec:
  # ...
  storage:
    azure:
      # ...
      networkAccess:
        type: Internal
        internal:
          subnetName: <subnet_name>
          vnetName: <vnet_name>
          networkResourceGroupName: <network_resource_group_name>
# ...
```

2. Optional: Enter the following command to confirm that the Operator has completed provisioning. This might take a few minutes.

```
$ oc get configs.imageregistry/cluster -o=jsonpath="{.spec.storage.azure.privateEndpointName}" -w
```



NOTE

When redirect is enabled, pulling images from outside of the cluster will not work.

Verification

1. Fetch the registry service name by running the following command:

```
$ oc get imagestream -n openshift
```

Example output

NAME	IMAGE REPOSITORY	TAGS	UPDATED
cli	image-registry.openshift-image-registry.svc:5000/openshift/cli	latest	8 hours ago
...			

2. Enter debug mode by running the following command:

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

3. Run the suggested **chroot** command. For example:

```
$ chroot /host
```

4. Enter the following command to log in to your container registry:

```
$ podman login --tls-verify=false -u unused -p $(oc whoami -t) image-registry.openshift-image-registry.svc:5000
```

Example output

```
Login Succeeded!
```

5. Enter the following command to verify that you can pull an image from the registry:

```
$ podman pull --tls-verify=false image-registry.openshift-image-registry.svc:5000/openshift/tools
```

Example output

```
Trying to pull image-registry.openshift-image-registry.svc:5000/openshift/tools/openshift/tools...
Getting image source signatures
Copying blob 6b245f040973 done
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
22667f53682a2920948d19c7133ab1c9c3f745805c14125859d20cede07f11f9
```

2.5.4. Optional: Disabling redirect when using a private storage endpoint on Azure

By default, redirect is enabled when using the image registry. Redirect allows off-loading of traffic from the registry pods into the object storage, which makes pull faster. When redirect is enabled and the storage account is private, users from outside of the cluster are unable to pull images from the registry.

In some cases, users might want to disable redirect so that users from outside of the cluster can pull images from the registry.

Use the following procedure to disable redirect.

Prerequisites

- You have configured the image registry to run on Azure.

- You have configured a route.

Procedure

- Enter the following command to disable redirect on the image registry configuration:

```
$ oc patch configs.imageregistry cluster --type=merge -p '{"spec":{"disableRedirect": true}}'
```

Verification

1. Fetch the registry service name by running the following command:

```
$ oc get imagestream -n openshift
```

Example output

```
NAME IMAGE REPOSITORY TAGS UPDATED
cli default-route-openshift-image-registry.<cluster_dns>/cli latest 8 hours ago
...
```

2. Enter the following command to log in to your container registry:

```
$ podman login --tls-verify=false -u unused -p $(oc whoami -t) default-route-openshift-image-registry.<cluster_dns>
```

Example output

```
Login Succeeded!
```

3. Enter the following command to verify that you can pull an image from the registry:

```
$ podman pull --tls-verify=false default-route-openshift-image-registry.<cluster_dns>/openshift/tools
```

Example output

```
Trying to pull default-route-openshift-image-registry.<cluster_dns>/openshift/tools...
Getting image source signatures
Copying blob 6b245f040973 done
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
22667f53682a2920948d19c7133ab1c9c3f745805c14125859d20cede07f11f9
```

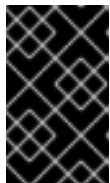

CHAPTER 3. CONFIGURING MULTI-ARCHITECTURE COMPUTE MACHINES ON AN OPENSIFT CLUSTER

3.1. ABOUT CLUSTERS WITH MULTI-ARCHITECTURE COMPUTE MACHINES

An OpenShift Container Platform cluster with multi-architecture compute machines is a cluster that supports compute machines with different architectures.

Configuring multi-architecture compute machines involves some additional considerations:

- When there are nodes with multiple architectures in your cluster, the architecture of the container image that you deploy to a node must be consistent with the architecture of that node. You need to ensure that the pod is assigned to the node with the appropriate architecture and that it matches the container image architecture. For more information on assigning pods to nodes, see [Assigning pods to nodes](#).
- In installer-provisioned installations, you are restricted to using the infrastructure provided by a single cloud provider. Adding external nodes, regardless of their architecture, to these clusters is not supported.
- Clusters that are installed with the platform type **none** are unable to use some features, such as managing compute machines with the Machine API. This limitation applies even if the compute machines that are attached to the cluster are installed on a platform that would normally support the feature. This parameter cannot be changed after installation.



IMPORTANT

Review the information in the [guidelines for deploying OpenShift Container Platform on non-tested platforms](#) before you attempt to install an OpenShift Container Platform cluster in virtualized or cloud environments.

- The Cluster Samples Operator is not supported on clusters with multi-architecture compute machines. Your cluster can be created without this capability. For more information, see [Cluster capabilities](#).
- For information on migrating your single-architecture cluster to a cluster that supports multi-architecture compute machines, see [Migrating to a cluster with multi-architecture compute machines](#).

3.1.1. Configuring your cluster with multi-architecture compute machines

To create a cluster with multi-architecture compute machines with different installation options and platforms, you can use the documentation in the following table:

Table 3.1. Cluster with multi-architecture compute machine installation options

Documentation section	Platform	User-provisioned installation	Installer-provisioned installation	Control Plane	Compute node
Creating a cluster with multi-architecture compute machines on Azure	Microsoft Azure	✓	✓	aarch64 or x86_64	aarch64 , x86_64
Creating a cluster with multi-architecture compute machines on AWS	Amazon Web Services (AWS)	✓	✓	aarch64 or x86_64	aarch64 , x86_64
Creating a cluster with multi-architecture compute machines on Google Cloud	Google Cloud		✓	aarch64 or x86_64	aarch64 , x86_64
Creating a cluster with multi-architecture compute machines on bare metal, IBM Power, or IBM Z	Bare metal	✓	✓	aarch64 or x86_64	aarch64 , x86_64
	IBM Power	✓		x86_64 or ppc64le	x86_64 , ppc64le
	IBM Z	✓		x86_64 or s390x	x86_64 , s390x
Creating a cluster with multi-architecture compute machines on IBM Z® and IBM® LinuxONE with z/VM	IBM Z® and IBM® LinuxONE	✓		x86_64	x86_64 , s390x
Creating a cluster with multi-architecture compute machines on IBM Z® and IBM® LinuxONE with RHEL KVM	IBM Z® and IBM® LinuxONE	✓		x86_64	x86_64 , s390x
Creating a cluster with multi-architecture compute machines on IBM Power®	IBM Power®	✓		x86_64	x86_64 , ppc64le



IMPORTANT

Autoscaling from zero is currently not supported on Google Cloud.

3.2. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINE ON AZURE

To deploy an Azure cluster with multi-architecture compute machines, you must first create a single-architecture Azure installer-provisioned cluster that uses the multi-architecture installer binary. For more information on Azure installations, see [Installing a cluster on Azure with customizations](#).

You can also migrate your current cluster with single-architecture compute machines to a cluster with multi-architecture compute machines. For more information, see [Migrating to a cluster with multi-architecture compute machines](#).

After creating a multi-architecture cluster, you can add nodes with different architectures to the cluster.

3.2.1. Verifying cluster compatibility

Before you can start adding compute nodes of different architectures to your cluster, you must verify that your cluster is multi-architecture compatible.

Prerequisites

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

Procedure

1. Log in to the OpenShift CLI (**oc**).
2. You can check that your cluster uses the architecture payload by running the following command:

```
$ oc adm release info -o jsonpath="{.metadata.metadata}"
```

Verification

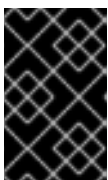
- If you see the following output, your cluster is using the multi-architecture payload:

```
{
  "release.openshift.io/architecture": "multi",
  "url": "https://access.redhat.com/errata/<errata_version>"
}
```

You can then begin adding multi-arch compute nodes to your cluster.

- If you see the following output, your cluster is not using the multi-architecture payload:

```
{
  "url": "https://access.redhat.com/errata/<errata_version>"
}
```



IMPORTANT

To migrate your cluster so the cluster supports multi-architecture compute machines, follow the procedure in [Migrating to a cluster with multi-architecture compute machines](#).

3.2.2. Creating a 64-bit ARM boot image using the Azure image gallery

The following procedure describes how to manually generate a 64-bit ARM boot image.

Prerequisites

- You installed the Azure CLI (**az**).
- You created a single-architecture Azure installer-provisioned cluster with the multi-architecture installer binary.

Procedure

1. Log in to your Azure account:

```
$ az login
```

2. Create a storage account and upload the **aarch64** virtual hard disk (VHD) to your storage account. The OpenShift Container Platform installation program creates a resource group, however, the boot image can also be uploaded to a custom named resource group:

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

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

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

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

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

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

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

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

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

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

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

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

- Copy the RHCOS VHD into the storage container:

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

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

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

Example output

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

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

- Create an image gallery using the following command:

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

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

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

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

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

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

–

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

10. Your **arm64** boot image is now generated. You can access the ID of your image with the following command:

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

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

Example resourceID

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

3.2.3. Creating a 64-bit x86 boot image using the Azure image gallery

The following procedure describes how to manually generate a 64-bit x86 boot image.

Prerequisites

- You installed the Azure CLI (**az**).
- You created a single-architecture Azure installer-provisioned cluster with the multi-architecture installer binary.

Procedure

1. Log in to your Azure account by running the following command:

```
$ az login
```

2. Create a storage account and upload the **x86_64** virtual hard disk (VHD) to your storage account by running the following command. The OpenShift Container Platform installation program creates a resource group. However, the boot image can also be uploaded to a custom named resource group:

```
$ az storage account create -n ${STORAGE_ACCOUNT_NAME} -g
${RESOURCE_GROUP} -l westus --sku Standard_LRS 1
```

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

3. Create a storage container using the storage account you generated by running the following command:

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

4. Use the OpenShift Container Platform installation program JSON file to extract the URL and **x86_64** VHD name:
 - a. Extract the **URL** field and set it to **RHCOS_VHD_ORIGIN_URL** as the file name by running the following command:

```
$ RHCOS_VHD_ORIGIN_URL=$(oc -n openshift-machine-config-operator get
configmap/coreos-bootimages -o jsonpath='{.data.stream}' | jq -r
'.architectures.x86_64."rhel-coreos-extensions"."azure-disk".url')
```

- b. Extract the **x86_64** VHD name and set it to **BLOB_NAME** as the file name by running the following command:

```
$ BLOB_NAME=rhcos-$(oc -n openshift-machine-config-operator get configmap/coreos-
bootimages -o jsonpath='{.data.stream}' | jq -r '.architectures.x86_64."rhel-coreos-
extensions"."azure-disk".release')-azure.x86_64.vhd
```

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

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

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

6. Copy the RHCOS VHD into the storage container by running the following command:

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

You can check the status of the copying process by running the following command:

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

Example output

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

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

7. Create an image gallery by running the following command:

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

8. Use the image gallery to create an image definition by running the following command:

```
$ az sig image-definition create --resource-group ${RESOURCE_GROUP} --gallery-name
${GALLERY_NAME} --gallery-image-definition rhcos-x86_64 --publisher RedHat --offer
x86_64 --sku x86_64 --os-type linux --architecture x64 --hyper-v-generation V2
```

In this example command, **rhcos-x86_64** is the name of the image definition.

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

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

10. Use the **RHCOS_VHD_URL** file, your storage account, resource group, and image gallery to create an image version by running the following command:

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

In this example, **1.0.0** is the image version.

11. Optional: Access the ID of the generated **x86_64** boot image by running the following command:

```
$ az sig image-version show -r $GALLERY_NAME -g $RESOURCE_GROUP -i rhcos-
x86_64 -e 1.0.0
```

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

Example resourceID

```
/resourceGroups/${RESOURCE_GROUP}/providers/Microsoft.Compute/galleries/${GALLERY
_NAME}/images/rhcos-x86_64/versions/1.0.0
```

3.2.4. Adding a multi-architecture compute machine set to your Azure cluster

After creating a multi-architecture cluster, you can add nodes with different architectures.

You can add multi-architecture compute machines to a multi-architecture cluster in the following ways:

- Adding 64-bit x86 compute machines to a cluster that uses 64-bit ARM control plane machines and already includes 64-bit ARM compute machines. In this case, 64-bit x86 is considered the secondary architecture.

- Adding 64-bit ARM compute machines to a cluster that uses 64-bit x86 control plane machines and already includes 64-bit x86 compute machines. In this case, 64-bit ARM is considered the secondary architecture.

To create a custom compute machine set on Azure, see "Creating a compute machine set on Azure".



NOTE

Before adding a secondary architecture node to your cluster, it is recommended to install the Multiarch Tuning Operator, and deploy a **ClusterPodPlacementConfig** custom resource. For more information, see "Managing workloads on multi-architecture clusters by using the Multiarch Tuning Operator".

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You created a 64-bit ARM or 64-bit x86 boot image.
- You used the installation program to create a 64-bit ARM or 64-bit x86 single-architecture Azure cluster with the multi-architecture installer binary.

Procedure

1. Log in to the OpenShift CLI (**oc**).
2. Create a YAML file, and add the configuration to create a compute machine set to control the 64-bit ARM or 64-bit x86 compute nodes in your cluster.

Example **MachineSet** object for an Azure 64-bit ARM or 64-bit x86 compute node

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

```

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

```

- ❶ Set the **resourceID** parameter to either **arm64** or **amd64** boot image.
- ❷ Set the **vmSize** parameter to the instance type used in your installation. Some example instance types are **Standard_D4ps_v5** or **D8ps**.

3. Create the compute machine set by running the following command:

```
$ oc create -f <file_name> ❶
```

- ❶ Replace **<file_name>** with the name of the YAML file with compute machine set configuration. For example: **arm64-machine-set-0.yaml**, or **amd64-machine-set-0.yaml**.

Verification

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

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

The output must include the machine set that you created.

Example output

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

2. You can check if the nodes are ready and schedulable by running the following command:

```
$ oc get nodes
```

Additional resources

- [Creating a compute machine set on Azure](#)
- [Managing workloads on multi-architecture clusters by using the Multiarch Tuning Operator](#)

3.3. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES ON AWS

To create an AWS cluster with multi-architecture compute machines, you must first create a single-architecture AWS installer-provisioned cluster with the multi-architecture installer binary. For more information on AWS installations, see [Installing a cluster on AWS with customizations](#).

You can also migrate your current cluster with single-architecture compute machines to a cluster with multi-architecture compute machines. For more information, see [Migrating to a cluster with multi-architecture compute machines](#).

After creating a multi-architecture cluster, you can add nodes with different architectures to the cluster.

3.3.1. Verifying cluster compatibility

Before you can start adding compute nodes of different architectures to your cluster, you must verify that your cluster is multi-architecture compatible.

Prerequisites

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

Procedure

1. Log in to the OpenShift CLI (**oc**).
2. You can check that your cluster uses the architecture payload by running the following command:

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

Verification

- If you see the following output, your cluster is using the multi-architecture payload:

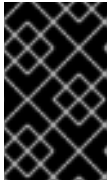
```
{
```

```
"release.openshift.io/architecture": "multi",
"url": "https://access.redhat.com/errata/<errata_version>"
}
```

You can then begin adding multi-arch compute nodes to your cluster.

- If you see the following output, your cluster is not using the multi-architecture payload:

```
{
"url": "https://access.redhat.com/errata/<errata_version>"
}
```



IMPORTANT

To migrate your cluster so the cluster supports multi-architecture compute machines, follow the procedure in [Migrating to a cluster with multi-architecture compute machines](#).

3.3.2. Adding a multi-architecture compute machine set to your AWS cluster

After creating a multi-architecture cluster, you can add nodes with different architectures.

You can add multi-architecture compute machines to a multi-architecture cluster in the following ways:

- Adding 64-bit x86 compute machines to a cluster that uses 64-bit ARM control plane machines and already includes 64-bit ARM compute machines. In this case, 64-bit x86 is considered the secondary architecture.
- Adding 64-bit ARM compute machines to a cluster that uses 64-bit x86 control plane machines and already includes 64-bit x86 compute machines. In this case, 64-bit ARM is considered the secondary architecture.



NOTE

Before adding a secondary architecture node to your cluster, it is recommended to install the Multiarch Tuning Operator, and deploy a **ClusterPodPlacementConfig** custom resource. For more information, see "Managing workloads on multi-architecture clusters by using the Multiarch Tuning Operator".

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You used the installation program to create an 64-bit ARM or 64-bit x86 single-architecture AWS cluster with the multi-architecture installer binary.

Procedure

1. Log in to the OpenShift CLI (**oc**).
2. Create a YAML file, and add the configuration to create a compute machine set to control the 64-bit ARM or 64-bit x86 compute nodes in your cluster.

Example MachineSet object for an AWS 64-bit ARM or x86 compute node

```

apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
  name: <infrastructure_id>-aws-machine-set-0 2
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id> 3
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>-<zone> 4
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: <role> 5
        machine.openshift.io/cluster-api-machine-type: <role> 6
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>-<zone> 7
    spec:
      metadata:
        labels:
          node-role.kubernetes.io/<role>: ""
      providerSpec:
        value:
          ami:
            id: ami-02a574449d4f4d280 8
          apiVersion: awsproviderconfig.openshift.io/v1beta1
          blockDevices:
            - ebs:
                iops: 0
                volumeSize: 120
                volumeType: gp2
          credentialsSecret:
            name: aws-cloud-credentials
          deviceIndex: 0
          iamInstanceProfile:
            id: <infrastructure_id>-worker-profile 9
          instanceType: m6g.xlarge 10
          kind: AWSMachineProviderConfig
          placement:
            availabilityZone: us-east-1a 11
            region: <region> 12
          securityGroups:
            - filters:
                - name: tag:Name
                  values:
                    - <infrastructure_id>-node 13
          subnet:
            filters:
              - name: tag:Name
                values:
                  - <infrastructure_id>-subnet-private-<zone>

```

```
tags:
  - name: kubernetes.io/cluster/<infrastructure_id> 14
    value: owned
  - name: <custom_tag_name>
    value: <custom_tag_value>
userDataSecret:
  name: worker-user-data
```

- 1 2 3 9 13 14 Specify the infrastructure ID that is based on the cluster ID that you set when you provisioned the cluster. If you have the OpenShift CLI (**oc**) installed, you can obtain the infrastructure ID by running the following command:

```
$ oc get -o jsonpath="{.status.infrastructureName}" infrastructure cluster
```

- 4 7 Specify the infrastructure ID, role node label, and zone.

- 5 6 Specify the role node label to add.

- 8 Specify a Red Hat Enterprise Linux CoreOS (RHCOS) Amazon Machine Image (AMI) for your AWS region for the nodes. The RHCOS AMI must be compatible with the machine architecture.

```
$ oc get configmap/coreos-bootimages \
  -n openshift-machine-config-operator \
  -o jsonpath='{.data.stream}' | jq \
  -r '.architectures.<arch>.images.aws.regions."<region>".image'
```

- 10 Specify a machine type that aligns with the CPU architecture of the chosen AMI. For more information, see "Tested instance types for AWS 64-bit ARM"
- 11 Specify the zone. For example, **us-east-1a**. Ensure that the zone you select has machines with the required architecture.
- 12 Specify the region. For example, **us-east-1**. Ensure that the zone you select has machines with the required architecture.

3. Create the compute machine set by running the following command:

```
$ oc create -f <file_name> 1
```

- 1 Replace **<file_name>** with the name of the YAML file with compute machine set configuration. For example: **aws-arm64-machine-set-0.yaml**, or **aws-amd64-machine-set-0.yaml**.

Verification

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

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

The output must include the machine set that you created.

Example output

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

2. You can check if the nodes are ready and schedulable by running the following command:

```
$ oc get nodes
```

Additional resources

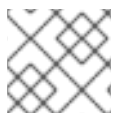
- [Tested instance types for AWS 64-bit ARM](#)
- [Managing workloads on multi-architecture clusters by using the Multiarch Tuning Operator](#)

3.4. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES ON GOOGLE CLOUD

To create a Google Cloud cluster with multi-architecture compute machines, you must first create a single-architecture Google Cloud installer-provisioned cluster with the multi-architecture installer binary. For more information on AWS installations, see [Installing a cluster on Google Cloud with customizations](#).

You can also migrate your current cluster with single-architecture compute machines to a cluster with multi-architecture compute machines. For more information, see [Migrating to a cluster with multi-architecture compute machines](#).

After creating a multi-architecture cluster, you can add nodes with different architectures to the cluster.



NOTE

Secure booting is currently not supported on 64-bit ARM machines for Google Cloud

3.4.1. Verifying cluster compatibility

Before you can start adding compute nodes of different architectures to your cluster, you must verify that your cluster is multi-architecture compatible.

Prerequisites

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

Procedure

1. Log in to the OpenShift CLI (**oc**).
2. You can check that your cluster uses the architecture payload by running the following command:

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

Verification

- If you see the following output, your cluster is using the multi-architecture payload:

```
{
  "release.openshift.io/architecture": "multi",
  "url": "https://access.redhat.com/errata/<errata_version>"
}
```

You can then begin adding multi-arch compute nodes to your cluster.

- If you see the following output, your cluster is not using the multi-architecture payload:

```
{
  "url": "https://access.redhat.com/errata/<errata_version>"
}
```



IMPORTANT

To migrate your cluster so the cluster supports multi-architecture compute machines, follow the procedure in [Migrating to a cluster with multi-architecture compute machines](#).

3.4.2. Adding a multi-architecture compute machine set to your Google Cloud cluster

After creating a multi-architecture cluster, you can add nodes with different architectures.

You can add multi-architecture compute machines to a multi-architecture cluster in the following ways:

- Adding 64-bit x86 compute machines to a cluster that uses 64-bit ARM control plane machines and already includes 64-bit ARM compute machines. In this case, 64-bit x86 is considered the secondary architecture.
- Adding 64-bit ARM compute machines to a cluster that uses 64-bit x86 control plane machines and already includes 64-bit x86 compute machines. In this case, 64-bit ARM is considered the secondary architecture.



NOTE

Before adding a secondary architecture node to your cluster, it is recommended to install the Multiarch Tuning Operator, and deploy a **ClusterPodPlacementConfig** custom resource. For more information, see "Managing workloads on multi-architecture clusters by using the Multiarch Tuning Operator".

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You used the installation program to create a 64-bit x86 or 64-bit ARM single-architecture Google Cloud cluster with the multi-architecture installer binary.

Procedure

1. Log in to the OpenShift CLI (**oc**).

2. Create a YAML file, and add the configuration to create a compute machine set to control the 64-bit ARM or 64-bit x86 compute nodes in your cluster.

Example **MachineSet** object for a Google Cloud 64-bit ARM or 64-bit x86 compute node

```

apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> ❶
  name: <infrastructure_id>-w-a
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-w-a
  template:
    metadata:
      creationTimestamp: null
      labels:
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: <role> ❷
        machine.openshift.io/cluster-api-machine-type: <role>
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-w-a
    spec:
      metadata:
        labels:
          node-role.kubernetes.io/<role>: ""
      providerSpec:
        value:
          apiVersion: gcpprovider.openshift.io/v1beta1
          canIPForward: false
          credentialsSecret:
            name: gcp-cloud-credentials
          deletionProtection: false
          disks:
            - autoDelete: true
              boot: true
              image: <path_to_image> ❸
              labels: null
              sizeGb: 128
              type: pd-ssd
          gcpMetadata: ❹
            - key: <custom_metadata_key>
              value: <custom_metadata_value>
          kind: GCPMachineProviderSpec
          machineType: n1-standard-4 ❺
          metadata:
            creationTimestamp: null
          networkInterfaces:
            - network: <infrastructure_id>-network
              subnet: <infrastructure_id>-worker-subnet

```

```

projectID: <project_name> 6
region: us-central1 7
serviceAccounts:
- email: <infrastructure_id>-w@<project_name>.iam.gserviceaccount.com
  scopes:
  - https://www.googleapis.com/auth/cloud-platform
tags:
- <infrastructure_id>-worker
userDataSecret:
  name: worker-user-data
zone: us-central1-a

```

- 1 Specify the infrastructure ID that is based on the cluster ID that you set when you provisioned the cluster. You can obtain the infrastructure ID by running the following command:

```
$ oc get -o jsonpath='{.status.infrastructureName}' infrastructure cluster
```

- 2 Specify the role node label to add.
- 3 Specify the path to the image that is used in current compute machine sets. You need the project and image name for your path to image.

To access the project and image name, run the following command:

```
$ oc get configmap/coreos-bootimages \
-n openshift-machine-config-operator \
-o jsonpath='{.data.stream}' | jq \
-r '.architectures.aarch64.images.gcp'
```

Example output

```

"gcp": {
  "release": "415.92.202309142014-0",
  "project": "rhcos-cloud",
  "name": "rhcos-415-92-202309142014-0-gcp-aarch64"
}

```

Use the **project** and **name** parameters from the output to create the path to image field in your machine set. The path to the image should follow the following format:

```
$ projects/<project>/global/images/<image_name>
```

- 4 Optional: Specify custom metadata in the form of a **key:value** pair. For example use cases, see the Google Cloud documentation for [setting custom metadata](#).
- 5 Specify a machine type that aligns with the CPU architecture of the chosen OS image. For more information, see "Tested instance types for Google Cloud on 64-bit ARM infrastructures".
- 6 Specify the name of the Google Cloud project that you use for your cluster.
- 7 Specify the region. For example, **us-central1**. Ensure that the zone you select has machines with the required architecture.

3. Create the compute machine set by running the following command:

```
$ oc create -f <file_name> 1
```

- 1 Replace **<file_name>** with the name of the YAML file with compute machine set configuration. For example: **gcp-arm64-machine-set-0.yaml**, or **gcp-amd64-machine-set-0.yaml**.

Verification

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

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

The output must include the machine set that you created.

Example output

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

2. You can check if the nodes are ready and schedulable by running the following command:

```
$ oc get nodes
```

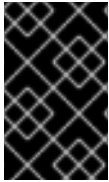
Additional resources

- [Tested instance types for Google Cloud on 64-bit ARM infrastructures](#)
- [Managing workloads on multi-architecture clusters by using the Multiarch Tuning Operator](#)

3.5. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES ON BARE METAL, IBM POWER, OR IBM Z

To create a cluster with multi-architecture compute machines on bare metal (**x86_64** or **aarch64**), IBM Power® (**ppc64le**), or IBM Z® (**s390x**) you must have an existing single-architecture cluster on one of these platforms. Follow the installations procedures for your platform:

- [Installing a user provisioned cluster on bare metal](#) . You can then add 64-bit ARM compute machines to your OpenShift Container Platform cluster on bare metal.
- [Installing a cluster on IBM Power®](#) . You can then add **x86_64** compute machines to your OpenShift Container Platform cluster on IBM Power®.
- [Installing a cluster on IBM Z® and IBM® LinuxONE](#) . You can then add **x86_64** compute machines to your OpenShift Container Platform cluster on IBM Z® and IBM® LinuxONE.



IMPORTANT

The bare metal installer-provisioned infrastructure and the Bare Metal Operator do not support adding secondary architecture nodes during the initial cluster setup. You can add secondary architecture nodes manually only after the initial cluster setup.

Before you can add additional compute nodes to your cluster, you must upgrade your cluster to one that uses the multi-architecture payload. For more information on migrating to the multi-architecture payload, see [Migrating to a cluster with multi-architecture compute machines](#).

The following procedures explain how to create a RHCOS compute machine using an ISO image or network PXE booting. This allows you to add additional nodes to your cluster and deploy a cluster with multi-architecture compute machines.



NOTE

Before adding a secondary architecture node to your cluster, it is recommended to install the Multiarch Tuning Operator, and deploy a **ClusterPodPlacementConfig** object. For more information, see [Managing workloads on multi-architecture clusters by using the Multiarch Tuning Operator](#).

3.5.1. Verifying cluster compatibility

Before you can start adding compute nodes of different architectures to your cluster, you must verify that your cluster is multi-architecture compatible.

Prerequisites

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

Procedure

1. Log in to the OpenShift CLI (**oc**).
2. You can check that your cluster uses the architecture payload by running the following command:

```
$ oc adm release info -o jsonpath='{ .metadata.metadata}'
```

Verification

- If you see the following output, your cluster is using the multi-architecture payload:

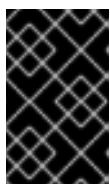
```
{
  "release.openshift.io/architecture": "multi",
  "url": "https://access.redhat.com/errata/<errata_version>"
}
```

You can then begin adding multi-arch compute nodes to your cluster.

- If you see the following output, your cluster is not using the multi-architecture payload:

```
{
  "url": "https://access.redhat.com/errata/<errata_version>"
}
```

}



IMPORTANT

To migrate your cluster so the cluster supports multi-architecture compute machines, follow the procedure in [Migrating to a cluster with multi-architecture compute machines](#).

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

Procedure

1. Extract the Ignition config file from the cluster by running the following command:

```
$ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --to=- > worker.ign
```

2. Upload the **worker.ign** Ignition config file you exported from your cluster to your HTTP server. Note the URLs of these files.
3. You can validate that the ignition files are available on the URLs. The following example gets the Ignition config files for the compute node:

```
$ curl -k http://<HTTP_server>/worker.ign
```

4. You can access the ISO image for booting your new machine by running to following command:

```
RHCOS_VHD_ORIGIN_URL=$(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o jsonpath='{.data.stream}' | jq -r '.architectures.<architecture>.artifacts.metal.formats.iso.disk.location')
```

5. 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.
6. Boot the RHCOS ISO image without specifying any options, or interrupting the live boot sequence. Wait for the installer to boot into a shell prompt in the RHCOS live environment.

**NOTE**

You can interrupt the RHCOS installation boot process to add kernel arguments. However, for this ISO procedure you must use the **coreos-installer** command as outlined in the following steps, instead of adding kernel arguments.

7. Run the **coreos-installer** command and specify the options that meet your installation requirements. At a minimum, you must specify the URL that points to the Ignition config file for the node type, and the device that you are installing to:

```
$ sudo coreos-installer install --ignition-url=http://<HTTP_server>/<node_type>.ign <device>
--ignition-hash=sha512-<digest> 1 2
```

- 1 You must run the **coreos-installer** command by using **sudo**, because the **core** user does not have the required root privileges to perform the installation.
- 2 The **--ignition-hash** option is required when the Ignition config file is obtained through an HTTP URL to validate the authenticity of the Ignition config file on the cluster node. **<digest>** is the Ignition config file SHA512 digest obtained in a preceding step.

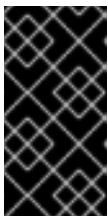
**NOTE**

If you want to provide your Ignition config files through an HTTPS server that uses TLS, you can add the internal certificate authority (CA) to the system trust store before running **coreos-installer**.

The following example initializes a compute node installation to the **/dev/sda** device. The Ignition config file for the compute node is obtained from an HTTP web server with the IP address 192.168.1.2:

```
$ sudo coreos-installer install --ignition-
url=http://192.168.1.2:80/installation_directory/worker.ign /dev/sda --ignition-hash=sha512-
a5a2d43879223273c9b60af66b44202a1d1248fc01cf156c46d4a79f552b6bad47bc8cc78ddf011
6e80c59d2ea9e32ba53bc807afbca581aa059311def2c3e3b
```

8. Monitor the progress of the RHCOS installation on the console of the machine.

**IMPORTANT**

Ensure that the installation is successful on each node before commencing with the OpenShift Container Platform installation. Observing the installation process can also help to determine the cause of RHCOS installation issues that might arise.

9. Continue to create more compute machines for your cluster.

3.5.3. Creating 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.



NOTE

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 (**x86_64** + **aarch64**):

```
kernel http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> initrd=main
```

```
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign 1 2
initrd --name main http://<HTTP_server>/rhcos-<version>-live-initramfs.
<architecture>.img 3
boot
```

- 1 Specify the 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.live.rootfs_url** parameter value is the location of the **rootfs** file, and the **coreos.inst.ignition_url** parameter value is the location of the worker Ignition config file.
- 2 If you use multiple NICs, specify a single interface in the **ip** option. For example, to use DHCP on a NIC that is named **eno1**, set **ip=eno1:dhcp**.
- 3 Specify the location of the **initramfs** file that you uploaded to your HTTP server.



NOTE

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?](#) and "Enabling the serial console for PXE and ISO installation" in the "Advanced RHCOS installation configuration" section.



NOTE

To network boot the CoreOS **kernel** on **aarch64** architecture, you need to use a version of iPXE build with the **IMAGE_GZIP** option enabled. See [IMAGE_GZIP option in iPXE](#).

- For PXE (with UEFI and GRUB as second stage) on **aarch64**:

```
menuentry 'Install CoreOS' {
  linux rhcos-<version>-live-kernel-<architecture>
  coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
  <architecture>.img coreos.inst.install_dev=/dev/sda
  coreos.inst.ignition_url=http://<HTTP_server>/worker.ign 1 2
  initrd rhcos-<version>-live-initramfs.<architecture>.img 3
}
```

- 1 Specify the locations of the RHCOS files that you uploaded to your HTTP/TFTP server. The **kernel** parameter value is the location of the **kernel** file on your TFTP server. The **coreos.live.rootfs_url** parameter value is the location of the **rootfs** file, and the **coreos.inst.ignition_url** parameter value is the location of the worker Ignition config file on your HTTP Server.
- 2 If you use multiple NICs, specify a single interface in the **ip** option. For example, to use DHCP on a NIC that is named **eno1**, set **ip=eno1:dhcp**.

3. For each machine, create a file named `cert/ocsp-response.html`.

3. Specify the location of the **initramfs** file that you uploaded to your TFTP server.

2. Use the PXE or iPXE infrastructure to create the required compute machines for your cluster.

3.5.4. 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	v1.33.4
master-1	Ready	master	63m	v1.33.4
master-2	Ready	master	64m	v1.33.4

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.

- 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
...
```

5. If the remaining CSRs are not approved, and are in the **Pending** status, approve the CSRs for your cluster machines:

- To approve them individually, run the following command for each valid CSR:

```
$ oc adm certificate approve <csr_name> 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:

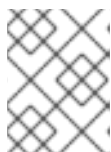
```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}{{end}}{{end}}' | xargs oc adm certificate approve
```

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  v1.33.4
master-1  Ready   master  73m  v1.33.4
master-2  Ready   master  74m  v1.33.4
worker-0  Ready   worker  11m  v1.33.4
worker-1  Ready   worker  11m  v1.33.4
```



NOTE

It can take a few minutes after approval of the server CSRs for the machines to transition to the **Ready** status.

Additional information

- [Certificate Signing Requests](#)

3.6. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES ON IBM Z AND IBM LINUXONE WITH Z/VM

To create a cluster with multi-architecture compute machines on IBM Z® and IBM® LinuxONE (s390x) with z/VM, you must have an existing single-architecture x86_64 cluster. You can then add s390x compute machines to your OpenShift Container Platform cluster.

Before you can add **s390x** nodes to your cluster, you must upgrade your cluster to one that uses the multi-architecture payload. For more information on migrating to the multi-architecture payload, see [Migrating to a cluster with multi-architecture compute machines](#).

The following procedures explain how to create a RHCOS compute machine using a z/VM instance. This will allow you to add **s390x** nodes to your cluster and deploy a cluster with multi-architecture compute machines.

To create an IBM Z® or IBM® LinuxONE (**s390x**) cluster with multi-architecture compute machines on **x86_64**, follow the instructions for [Installing a cluster on IBM Z® and IBM® LinuxONE](#). You can then add **x86_64** compute machines as described in [Creating a cluster with multi-architecture compute machines on bare metal, IBM Power, or IBM Z](#).



NOTE

Before adding a secondary architecture node to your cluster, it is recommended to install the Multiarch Tuning Operator, and deploy a **ClusterPodPlacementConfig** object. For more information, see [Managing workloads on multi-architecture clusters by using the Multiarch Tuning Operator](#).

3.6.1. Verifying cluster compatibility

Before you can start adding compute nodes of different architectures to your cluster, you must verify that your cluster is multi-architecture compatible.

Prerequisites

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

Procedure

- Log in to the OpenShift CLI (**oc**).
- You can check that your cluster uses the architecture payload by running the following command:

```
$ oc adm release info -o jsonpath="{.metadata.metadata}"
```

Verification

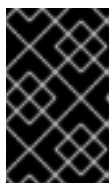
- If you see the following output, your cluster is using the multi-architecture payload:

```
{
  "release.openshift.io/architecture": "multi",
  "url": "https://access.redhat.com/errata/<errata_version>"
}
```

You can then begin adding multi-arch compute nodes to your cluster.

- If you see the following output, your cluster is not using the multi-architecture payload:

```
{
  "url": "https://access.redhat.com/errata/<errata_version>"
}
```

**IMPORTANT**

To migrate your cluster so the cluster supports multi-architecture compute machines, follow the procedure in [Migrating to a cluster with multi-architecture compute machines](#).

3.6.2. Creating RHCOS machines on IBM Z with z/VM

You can create more Red Hat Enterprise Linux CoreOS (RHCOS) compute machines running on IBM Z® with z/VM and attach them to your existing cluster.

Prerequisites

- You have a domain name server (DNS) that can perform hostname and reverse lookup for the nodes.
- You have an HTTP or HTTPS server running on your provisioning machine that is accessible to the machines you create.

Procedure

1. Extract the Ignition config file from the cluster by running the following command:

```
$ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --to=- > worker.ign
```

2. Upload the **worker.ign** Ignition config file you exported from your cluster to your HTTP server. Note the URL of this file.
3. You can validate that the Ignition file is available on the URL. The following example gets the Ignition config file for the compute node:

```
$ curl -k http://<http_server>/worker.ign
```

4. Download the RHEL live **kernel**, **initramfs**, and **rootfs** files by running the following commands:

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.kernel.location')
```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.initramfs.location')
```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.rootfs.location')
```

5. Move the downloaded RHEL live **kernel**, **initramfs**, and **rootfs** files to an HTTP or HTTPS server that is accessible from the RHCOS guest you want to add.

6. Create a parameter file for the guest. The following parameters are specific for the virtual machine:

- Optional: To specify a static IP address, add an **ip=** parameter with the following entries, with each separated by a colon:
 - i. The IP address for the machine.
 - ii. An empty string.
 - iii. The gateway.
 - iv. The netmask.
 - v. The machine host and domain name in the form **hostname.domainname**. If you omit this value, RHCOS obtains the hostname through a reverse DNS lookup.
 - vi. The network interface name. If you omit this value, RHCOS applies the IP configuration to all available interfaces.
 - vii. The value **none**.
 - For **coreos.inst.ignition_url=**, specify the URL to the **worker.ign** file. Only HTTP and HTTPS protocols are supported.
 - For **coreos.live.rootfs_url=**, specify the matching rootfs artifact for the **kernel** and **initramfs** you are booting. Only HTTP and HTTPS protocols are supported.
 - For installations on DASD-type disks, complete the following tasks:
 - i. For **coreos.inst.install_dev=**, specify **/dev/dasda**.
 - ii. Use **rd.dasd=** to specify the DASD where RHCOS is to be installed.
 - iii. You can adjust further parameters if required.
- The following is an example parameter file, **additional-worker-dasd.parm**:

```

cio_ignore=all,!condev rd.neednet=1 \
console=ttysclp0 \
coreos.inst.install_dev=/dev/dasda \
coreos.inst.ignition_url=http://<http_server>/worker.ign \
coreos.live.rootfs_url=http://<http_server>/rhcos-<version>-live-rootfs.
<architecture>.img \
ip=<ip>::<gateway>:<netmask>:<hostname>::none nameserver=<dns> \
rd.znet=qeth,0.0.bdf0,0.0.bdf1,0.0.bdf2,layer2=1,portno=0 \
rd.dasd=0.0.3490 \
zfcp.allow_lun_scan=0

```

Write all options in the parameter file as a single line and make sure that you have no newline characters.

- For installations on FCP-type disks, complete the following tasks:
 - i. Use **rd.zfcp=<adapter>,<wwpn>,<lun>** to specify the FCP disk where RHCOS is to be installed. For multipathing, repeat this step for each additional path.

**NOTE**

When you install with multiple paths, you must enable multipathing directly after the installation, not at a later point in time, as this can cause problems.

- ii. Set the install device as: **coreos.inst.install_dev=/dev/sda**.

**NOTE**

If additional LUNs are configured with NPIV, FCP requires **zfcplib.allow_lun_scan=0**. If you must enable **zfcplib.allow_lun_scan=1** because you use a CSI driver, for example, you must configure your NPIV so that each node cannot access the boot partition of another node.

- iii. You can adjust further parameters if required.

**IMPORTANT**

Additional postinstallation steps are required to fully enable multipathing. For more information, see "Enabling multipathing with kernel arguments on RHCOS" in *Machine configuration*.

The following is an example parameter file, **additional-worker-fcp.parm** for a worker node with multipathing:

```

cio_ignore=all,!condev rd.neednet=1 \
console=ttysclp0 \
coreos.inst.install_dev=/dev/sda \
coreos.live.rootfs_url=http://<http_server>/rhcos-<version>-live-rootfs.
<architecture>.img \
coreos.inst.ignition_url=http://<http_server>/worker.ign \
ip=<ip>::<gateway>:<netmask>:<hostname>::none nameserver=<dns> \
rd.znet=qeth,0.0.bdf0,0.0.bdf1,0.0.bdf2,layer2=1,portno=0 \
zfcplib.allow_lun_scan=0 \
rd.zfcplib=0.0.1987,0x50050763070bc5e3,0x4008400B00000000 \
rd.zfcplib=0.0.19C7,0x50050763070bc5e3,0x4008400B00000000 \
rd.zfcplib=0.0.1987,0x50050763071bc5e3,0x4008400B00000000 \
rd.zfcplib=0.0.19C7,0x50050763071bc5e3,0x4008400B00000000

```

Write all options in the parameter file as a single line and make sure that you have no newline characters.

7. Transfer the **initramfs**, **kernel**, parameter files, and RHCOS images to z/VM, for example, by using FTP. For details about how to transfer the files with FTP and boot from the virtual reader, see [Booting the installation on IBM Z® to install RHEL in z/VM](#) .
8. Punch the files to the virtual reader of the z/VM guest virtual machine. See [PUNCH](#) in IBM® Documentation.

TIP

You can use the CP PUNCH command or, if you use Linux, the **vmur** command to transfer files between two z/VM guest virtual machines.

9. Log in to CMS on the bootstrap machine.
10. IPL the bootstrap machine from the reader by running the following command:

```
$ ipl c
```

See [IPL](#) in IBM® Documentation.

3.6.3. 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   v1.33.4
master-1  Ready    master   63m   v1.33.4
master-2  Ready    master   64m   v1.33.4
```

The output lists all of the machines that you created.



NOTE

The preceding output might not include the compute nodes, also known as worker nodes, until some CSRs are approved.

2. Review the pending CSRs and ensure that you see the client requests with the **Pending** or **Approved** status for each machine that you added to the cluster:

```
$ oc get csr
```

Example output

```
NAME      AGE   REQUESTOR                                CONDITION
csr-8b2br  15m   system:serviceaccount:openshift-machine-config-operator:node-
```



```

bootstrapper Pending
csr-8vnps 15m system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper Pending
...

```

In this example, two machines are joining the cluster. You might see more approved CSRs in the list.

3. If the CSRs were not approved, after all of the pending CSRs for the machines you added are in **Pending** status, approve the CSRs for your cluster machines:



NOTE

Because the CSRs rotate automatically, approve your CSRs within an hour of adding the machines to the cluster. If you do not approve them within an hour, the certificates will rotate, and more than two certificates will be present for each node. You must approve all of these certificates. After the client CSR is approved, the Kubelet creates a secondary CSR for the serving certificate, which requires manual approval. Then, subsequent serving certificate renewal requests are automatically approved by the **machine-approver** if the Kubelet requests a new certificate with identical parameters.



NOTE

For clusters running on platforms that are not machine API enabled, such as bare metal and other user-provisioned infrastructure, you must implement a method of automatically approving the kubelet serving certificate requests (CSRs). If a request is not approved, then the **oc exec**, **oc rsh**, and **oc logs** commands cannot succeed, because a serving certificate is required when the API server connects to the kubelet. Any operation that contacts the Kubelet endpoint requires this certificate approval to be in place. The method must watch for new CSRs, confirm that the CSR was submitted by the **node-bootstrapper** service account in the **system:node** or **system:admin** groups, and confirm the identity of the node.

- To approve them individually, run the following command for each valid CSR:

```
$ oc adm certificate approve <csr_name> 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.

4. Now that your client requests are approved, you must review the server requests for each machine that you added to the cluster:

```
$ oc get csr
```

Example output

```
NAME      AGE   REQUESTOR                                CONDITION
csr-bfd72 5m26s system:node:ip-10-0-50-126.us-east-2.compute.internal
Pending
csr-c57lv 5m26s system:node:ip-10-0-95-157.us-east-2.compute.internal
Pending
...
```

- 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
```

- 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   v1.33.4
master-1  Ready    master   73m   v1.33.4
master-2  Ready    master   74m   v1.33.4
worker-0  Ready    worker   11m   v1.33.4
worker-1  Ready    worker   11m   v1.33.4
```



NOTE

It can take a few minutes after approval of the server CSRs for the machines to transition to the **Ready** status.

Additional information

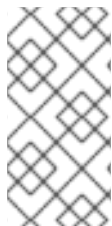
- [Certificate Signing Requests](#)

3.7. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES ON IBM Z AND IBM LINUXONE IN AN LPAR

To create a cluster with multi-architecture compute machines on IBM Z® and IBM® LinuxONE (**s390x**) in an LPAR, you must have an existing single-architecture **x86_64** cluster. You can then add **s390x** compute machines to your OpenShift Container Platform cluster.

Before you can add **s390x** nodes to your cluster, you must upgrade your cluster to one that uses the multi-architecture payload. For more information on migrating to the multi-architecture payload, see [Migrating to a cluster with multi-architecture compute machines](#).

The following procedures explain how to create a RHCOS compute machine using an LPAR instance. This will allow you to add **s390x** nodes to your cluster and deploy a cluster with multi-architecture compute machines.



NOTE

To create an IBM Z® or IBM® LinuxONE (**s390x**) cluster with multi-architecture compute machines on **x86_64**, follow the instructions for [Installing a cluster on IBM Z® and IBM® LinuxONE](#). You can then add **x86_64** compute machines as described in [Creating a cluster with multi-architecture compute machines on bare metal, IBM Power, or IBM Z](#).

3.7.1. Verifying cluster compatibility

Before you can start adding compute nodes of different architectures to your cluster, you must verify that your cluster is multi-architecture compatible.

Prerequisites

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

Procedure

- Log in to the OpenShift CLI (**oc**).
- You can check that your cluster uses the architecture payload by running the following command:

```
$ oc adm release info -o jsonpath="{.metadata.metadata}"
```

Verification

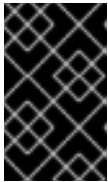
- If you see the following output, your cluster is using the multi-architecture payload:

```
{
  "release.openshift.io/architecture": "multi",
  "url": "https://access.redhat.com/errata/<errata_version>"
}
```

You can then begin adding multi-arch compute nodes to your cluster.

- If you see the following output, your cluster is not using the multi-architecture payload:

```
{
  "url": "https://access.redhat.com/errata/<errata_version>"
}
```



IMPORTANT

To migrate your cluster so the cluster supports multi-architecture compute machines, follow the procedure in [Migrating to a cluster with multi-architecture compute machines](#).

3.7.2. Creating RHCOS machines on IBM Z in an LPAR

You can create more Red Hat Enterprise Linux CoreOS (RHCOS) compute machines running on IBM Z® in a logical partition (LPAR) and attach them to your existing cluster.

Prerequisites

- You have a domain name server (DNS) that can perform hostname and reverse lookup for the nodes.
- You have an HTTP or HTTPS server running on your provisioning machine that is accessible to the machines you create.

Procedure

1. Extract the Ignition config file from the cluster by running the following command:

```
$ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --to=- > worker.ign
```

2. Upload the **worker.ign** Ignition config file you exported from your cluster to your HTTP server. Note the URL of this file.
3. You can validate that the Ignition file is available on the URL. The following example gets the Ignition config file for the compute node:

```
$ curl -k http://<http_server>/worker.ign
```

4. Download the RHEL live **kernel**, **initramfs**, and **rootfs** files by running the following commands:

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.kernel.location')
```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.initramfs.location')
```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.rootfs.location')
```

5. Move the downloaded RHEL live **kernel**, **initramfs**, and **rootfs** files to an HTTP or HTTPS server that is accessible from the RHCOS guest you want to add.
6. Create a parameter file for the guest. The following parameters are specific for the virtual machine:

- Optional: To specify a static IP address, add an **ip=** parameter with the following entries, with each separated by a colon:
 - i. The IP address for the machine.
 - ii. An empty string.
 - iii. The gateway.
 - iv. The netmask.
 - v. The machine host and domain name in the form **hostname.domainname**. If you omit this value, RHCOS obtains the hostname through a reverse DNS lookup.
 - vi. The network interface name. If you omit this value, RHCOS applies the IP configuration to all available interfaces.
 - vii. The value **none**.
- For **coreos.inst.ignition_url=**, specify the URL to the **worker.ign** file. Only HTTP and HTTPS protocols are supported.
- For **coreos.live.rootfs_url=**, specify the matching rootfs artifact for the **kernel** and **initramfs** you are booting. Only HTTP and HTTPS protocols are supported.
- For installations on DASD-type disks, complete the following tasks:
 - i. For **coreos.inst.install_dev=**, specify **/dev/dasda**.
 - ii. Use **rd.dasd=** to specify the DASD where RHCOS is to be installed.
 - iii. You can adjust further parameters if required.

The following is an example parameter file, **additional-worker-dasd.parm**:

```

cio_ignore=all,!condev rd.neednet=1 \
console=ttysclp0 \
coreos.inst.install_dev=/dev/dasda \
coreos.inst.ignition_url=http://<http_server>/worker.ign \
coreos.live.rootfs_url=http://<http_server>/rhcos-<version>-live-rootfs.
<architecture>.img \
ip=<ip>::<gateway>:<netmask>:<hostname>::none nameserver=<dns> \
rd.znet=qeth,0.0.bdf0,0.0.bdf1,0.0.bdf2,layer2=1,portno=0 \
rd.dasd=0.0.3490 \
zfcp.allow_lun_scan=0

```

Write all options in the parameter file as a single line and make sure that you have no newline characters.

- For installations on FCP-type disks, complete the following tasks:
 - i. Use **rd.zfcp=<adapter>,<wwpn>,<lun>** to specify the FCP disk where RHCOS is to be installed. For multipathing, repeat this step for each additional path.

**NOTE**

When you install with multiple paths, you must enable multipathing directly after the installation, not at a later point in time, as this can cause problems.

- ii. Set the install device as: **coreos.inst.install_dev=/dev/sda**.

**NOTE**

If additional LUNs are configured with NPIV, FCP requires **zfcplib.allow_lun_scan=0**. If you must enable **zfcplib.allow_lun_scan=1** because you use a CSI driver, for example, you must configure your NPIV so that each node cannot access the boot partition of another node.

- iii. You can adjust further parameters if required.

**IMPORTANT**

Additional postinstallation steps are required to fully enable multipathing. For more information, see "Enabling multipathing with kernel arguments on RHCOS" in *Machine configuration*.

The following is an example parameter file, **additional-worker-fcp.parm** for a worker node with multipathing:

```
cio_ignore=all,!condev rd.neednet=1 \
console=ttysclp0 \
coreos.inst.install_dev=/dev/sda \
coreos.live.rootfs_url=http://<http_server>/rhcos-<version>-live-rootfs.
<architecture>.img \
coreos.inst.ignition_url=http://<http_server>/worker.ign \
ip=<ip>::<gateway>:<netmask>:<hostname>::none nameserver=<dns> \
rd.znet=qeth,0.0.bdf0,0.0.bdf1,0.0.bdf2,layer2=1,portno=0 \
zfcplib.allow_lun_scan=0 \
rd.zfcplib=0.0.1987,0x50050763070bc5e3,0x4008400B00000000 \
rd.zfcplib=0.0.19C7,0x50050763070bc5e3,0x4008400B00000000 \
rd.zfcplib=0.0.1987,0x50050763071bc5e3,0x4008400B00000000 \
rd.zfcplib=0.0.19C7,0x50050763071bc5e3,0x4008400B00000000
```

Write all options in the parameter file as a single line and make sure that you have no newline characters.

7. Transfer the initramfs, kernel, parameter files, and RHCOS images to the LPAR, for example with FTP. For details about how to transfer the files with FTP and boot, see [Booting the installation on IBM Z® to install RHEL in an LPAR](#).
8. Boot the machine

3.7.3. 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   v1.33.4
master-1  Ready    master   63m   v1.33.4
master-2  Ready    master   64m   v1.33.4
```

The output lists all of the machines that you created.



NOTE

The preceding output might not include the compute nodes, also known as worker nodes, until some CSRs are approved.

2. Review the pending CSRs and ensure that you see the client requests with the **Pending** or **Approved** status for each machine that you added to the cluster:

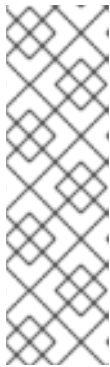
```
$ oc get csr
```

Example output

```
NAME      AGE   REQUESTOR                                     CONDITION
csr-8b2br  15m   system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper  Pending
csr-8vnps  15m   system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper  Pending
...
```

In this example, two machines are joining the cluster. You might see more approved CSRs in the list.

3. If the CSRs were not approved, after all of the pending CSRs for the machines you added are in **Pending** status, approve the CSRs for your cluster machines:

**NOTE**

Because the CSRs rotate automatically, approve your CSRs within an hour of adding the machines to the cluster. If you do not approve them within an hour, the certificates will rotate, and more than two certificates will be present for each node. You must approve all of these certificates. After the client CSR is approved, the Kubelet creates a secondary CSR for the serving certificate, which requires manual approval. Then, subsequent serving certificate renewal requests are automatically approved by the **machine-approver** if the Kubelet requests a new certificate with identical parameters.

**NOTE**

For clusters running on platforms that are not machine API enabled, such as bare metal and other user-provisioned infrastructure, you must implement a method of automatically approving the kubelet serving certificate requests (CSRs). If a request is not approved, then the **oc exec**, **oc rsh**, and **oc logs** commands cannot succeed, because a serving certificate is required when the API server connects to the kubelet. Any operation that contacts the Kubelet endpoint requires this certificate approval to be in place. The method must watch for new CSRs, confirm that the CSR was submitted by the **node-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
...			

5. If the remaining CSRs are not approved, and are in the **Pending** status, approve the CSRs for your cluster machines:

- To approve them individually, run the following command for each valid CSR:

```
$ oc adm certificate approve <csr_name> 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   v1.33.4
master-1  Ready    master   73m   v1.33.4
master-2  Ready    master   74m   v1.33.4
worker-0  Ready    worker   11m   v1.33.4
worker-1  Ready    worker   11m   v1.33.4
```



NOTE

It can take a few minutes after approval of the server CSRs for the machines to transition to the **Ready** status.

Additional information

- [Certificate Signing Requests](#)

3.8. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES ON IBM Z AND IBM LINUXONE WITH RHEL KVM

To create a cluster with multi-architecture compute machines on IBM Z® and IBM® LinuxONE (**s390x**) with RHEL KVM, you must have an existing single-architecture **x86_64** cluster. You can then add **s390x** compute machines to your OpenShift Container Platform cluster.

Before you can add **s390x** nodes to your cluster, you must upgrade your cluster to one that uses the multi-architecture payload. For more information on migrating to the multi-architecture payload, see [Migrating to a cluster with multi-architecture compute machines](#).

The following procedures explain how to create a RHCOS compute machine using a RHEL KVM instance. This will allow you to add **s390x** nodes to your cluster and deploy a cluster with multi-architecture compute machines.

To create an IBM Z® or IBM® LinuxONE (**s390x**) cluster with multi-architecture compute machines on **x86_64**, follow the instructions for [Installing a cluster on IBM Z® and IBM® LinuxONE](#) . You can then add **x86_64** compute machines as described in [Creating a cluster with multi-architecture compute machines on bare metal, IBM Power, or IBM Z](#).



NOTE

Before adding a secondary architecture node to your cluster, it is recommended to install the Multiarch Tuning Operator, and deploy a **ClusterPodPlacementConfig** object. For more information, see [Managing workloads on multi-architecture clusters by using the Multiarch Tuning Operator](#).

3.8.1. Verifying cluster compatibility

Before you can start adding compute nodes of different architectures to your cluster, you must verify that your cluster is multi-architecture compatible.

Prerequisites

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

Procedure

1. Log in to the OpenShift CLI (**oc**).
2. You can check that your cluster uses the architecture payload by running the following command:

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

Verification

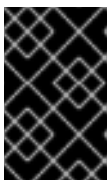
- If you see the following output, your cluster is using the multi-architecture payload:

```
{
  "release.openshift.io/architecture": "multi",
  "url": "https://access.redhat.com/errata/<errata_version>"
}
```

You can then begin adding multi-arch compute nodes to your cluster.

- If you see the following output, your cluster is not using the multi-architecture payload:

```
{
  "url": "https://access.redhat.com/errata/<errata_version>"
}
```



IMPORTANT

To migrate your cluster so the cluster supports multi-architecture compute machines, follow the procedure in [Migrating to a cluster with multi-architecture compute machines](#).

3.8.2. Creating RHCOS machines using **virt-install**

You can create more Red Hat Enterprise Linux CoreOS (RHCOS) compute machines for your cluster by using **virt-install**.

Prerequisites

- You have at least one LPAR running on RHEL 8.7 or later with KVM, referred to as RHEL KVM host in this procedure.
- The KVM/QEMU hypervisor is installed on the RHEL KVM host.
- You have a domain name server (DNS) that can perform hostname and reverse lookup for the nodes.
- An HTTP or HTTPS server is set up.

Procedure

1. Extract the Ignition config file from the cluster by running the following command:

```
$ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --to=- > worker.ign
```

2. Upload the **worker.ign** Ignition config file you exported from your cluster to your HTTP server. Note the URL of this file.
3. You can validate that the Ignition file is available on the URL. The following example gets the Ignition config file for the compute node:

```
$ curl -k http://<HTTP_server>/worker.ign
```

4. Download the RHEL live **kernel**, **initramfs**, and **rootfs** files by running the following commands:

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.kernel.location')
```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.initramfs.location')
```

```
$ curl -LO $(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o jsonpath='{.data.stream}' \
| jq -r '.architectures.s390x.artifacts.metal.formats.pxe.rootfs.location')
```

5. Move the downloaded RHEL live **kernel**, **initramfs**, and **rootfs** files to an HTTP or HTTPS server before you launch **virt-install**.
6. Create the new KVM guest nodes using the RHEL **kernel**, **initramfs**, and Ignition files; the new disk image; and adjusted parm line arguments.

```
$ virt-install \
```

```

--connect qemu:///system \
--name <vm_name> \
--autostart \
--os-variant rhel9.4 \ ❶
--cpu host \
--vcpus <vcpus> \
--memory <memory_mb> \
--disk <vm_name>.qcow2,size=<image_size> \
--network network=<virt_network_parm> \
--location <media_location>,kernel=<rhcos_kernel>,initrd=<rhcos_initrd> \ ❷
--extra-args "rd.neednet=1" \
--extra-args "coreos.inst.install_dev=/dev/vda" \
--extra-args "coreos.inst.ignition_url=http://<http_server>/worker.ign " \ ❸
--extra-args "coreos.live.rootfs_url=http://<http_server>/rhcos-<version>-live-rootfs.
<architecture>.img" \ ❹
--extra-args "ip=<ip>::<gateway>:<netmask>:<hostname>::none" \ ❺
--extra-args "nameserver=<dns>" \
--extra-args "console=ttysclp0" \
--noautoconsole \
--wait

```

- ❶ For **os-variant**, specify the RHEL version for the RHCOS compute machine. **rhel9.4** is the recommended version. To query the supported RHEL version of your operating system, run the following command:

```
$ osinfo-query os -f short-id
```



NOTE

The **os-variant** is case sensitive.

- ❷ For **--location**, specify the location of the kernel/initrd on the HTTP or HTTPS server.
- ❸ Specify the location of the **worker.ign** config file. Only HTTP and HTTPS protocols are supported.
- ❹ Specify the location of the **rootfs** artifact for the **kernel** and **initramfs** you are booting. Only HTTP and HTTPS protocols are supported.
- ❺ Optional: For **hostname**, specify the fully qualified hostname of the client machine.



NOTE

If you are using HAProxy as a load balancer, update your HAProxy rules for **ingress-router-443** and **ingress-router-80** in the **/etc/haproxy/haproxy.cfg** configuration file.

7. Continue to create more compute machines for your cluster.

3.8.3. 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   v1.33.4
master-1  Ready    master   63m   v1.33.4
master-2  Ready    master   64m   v1.33.4
```

The output lists all of the machines that you created.



NOTE

The preceding output might not include the compute nodes, also known as worker nodes, until some CSRs are approved.

2. Review the pending CSRs and ensure that you see the client requests with the **Pending** or **Approved** status for each machine that you added to the cluster:

```
$ oc get csr
```

Example output

```
NAME      AGE   REQUESTOR                                     CONDITION
csr-8b2br  15m   system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper  Pending
csr-8vnps  15m   system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper  Pending
...
```

In this example, two machines are joining the cluster. You might see more approved CSRs in the list.

3. If the CSRs were not approved, after all of the pending CSRs for the machines you added are in **Pending** status, approve the CSRs for your cluster machines:

**NOTE**

Because the CSRs rotate automatically, approve your CSRs within an hour of adding the machines to the cluster. If you do not approve them within an hour, the certificates will rotate, and more than two certificates will be present for each node. You must approve all of these certificates. After the client CSR is approved, the Kubelet creates a secondary CSR for the serving certificate, which requires manual approval. Then, subsequent serving certificate renewal requests are automatically approved by the **machine-approver** if the Kubelet requests a new certificate with identical parameters.

**NOTE**

For clusters running on platforms that are not machine API enabled, such as bare metal and other user-provisioned infrastructure, you must implement a method of automatically approving the kubelet serving certificate requests (CSRs). If a request is not approved, then the **oc exec**, **oc rsh**, and **oc logs** commands cannot succeed, because a serving certificate is required when the API server connects to the kubelet. Any operation that contacts the Kubelet endpoint requires this certificate approval to be in place. The method must watch for new CSRs, confirm that the CSR was submitted by the **node-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
...
```

5. If the remaining CSRs are not approved, and are in the **Pending** status, approve the CSRs for your cluster machines:

- To approve them individually, run the following command for each valid CSR:

```
$ oc adm certificate approve <csr_name> 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   v1.33.4
master-1  Ready    master   73m   v1.33.4
master-2  Ready    master   74m   v1.33.4
worker-0  Ready    worker   11m   v1.33.4
worker-1  Ready    worker   11m   v1.33.4
```



NOTE

It can take a few minutes after approval of the server CSRs for the machines to transition to the **Ready** status.

Additional information

- [Certificate Signing Requests](#)

3.9. CREATING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES ON IBM POWER

To create a cluster with multi-architecture compute machines on IBM Power® (**ppc64le**), you must have an existing single-architecture (**x86_64**) cluster. You can then add **ppc64le** compute machines to your OpenShift Container Platform cluster.



IMPORTANT

Before you can add **ppc64le** nodes to your cluster, you must upgrade your cluster to one that uses the multi-architecture payload. For more information on migrating to the multi-architecture payload, see [Migrating to a cluster with multi-architecture compute machines](#).

The following procedures explain how to create a RHCOS compute machine using an ISO image or network PXE booting. This will allow you to add **ppc64le** nodes to your cluster and deploy a cluster with multi-architecture compute machines.

To create an IBM Power® (**ppc64le**) cluster with multi-architecture compute machines on **x86_64**, follow the instructions for [Installing a cluster on IBM Power®](#). You can then add **x86_64** compute machines as described in [Creating a cluster with multi-architecture compute machines on bare metal, IBM Power, or IBM Z](#).



NOTE

Before adding a secondary architecture node to your cluster, it is recommended to install the Multiarch Tuning Operator, and deploy a **ClusterPodPlacementConfig** object. For more information, see [Managing workloads on multi-architecture clusters by using the Multiarch Tuning Operator](#).

3.9.1. Verifying cluster compatibility

Before you can start adding compute nodes of different architectures to your cluster, you must verify that your cluster is multi-architecture compatible.

Prerequisites

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



NOTE

When using multiple architectures, hosts for OpenShift Container Platform nodes must share the same storage layer. If they do not have the same storage layer, use a storage provider such as **nfs-provisioner**.



NOTE

You should limit the number of network hops between the compute and control plane as much as possible.

Procedure

- Log in to the OpenShift CLI (**oc**).
- You can check that your cluster uses the architecture payload by running the following command:

```
$ oc adm release info -o jsonpath="{ .metadata.metadata}"
```

Verification

- If you see the following output, your cluster is using the multi-architecture payload:

```
{
  "release.openshift.io/architecture": "multi",
  "url": "https://access.redhat.com/errata/<errata_version>"
}
```


You can then begin adding multi-arch compute nodes to your cluster.

- If you see the following output, your cluster is not using the multi-architecture payload:

```
{
  "url": "https://access.redhat.com/errata/<errata_version>"
}
```



IMPORTANT

To migrate your cluster so the cluster supports multi-architecture compute machines, follow the procedure in [Migrating to a cluster with multi-architecture compute machines](#).

3.9.2. Creating RHCOS machines using an ISO image

You can create more Red Hat Enterprise Linux CoreOS (RHCOS) compute machines for your 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.
- You must have the OpenShift CLI (**oc**) installed.

Procedure

1. Extract the Ignition config file from the cluster by running the following command:

```
$ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --to=- > worker.ign
```

2. Upload the **worker.ign** Ignition config file you exported from your cluster to your HTTP server. Note the URLs of these files.
3. You can validate that the ignition files are available on the URLs. The following example gets the Ignition config files for the compute node:

```
$ curl -k http://<HTTP_server>/worker.ign
```

4. You can access the ISO image for booting your new machine by running to following command:

```
RHCOS_VHD_ORIGIN_URL=$(oc -n openshift-machine-config-operator get configmap/coreos-bootimages -o jsonpath='{.data.stream}' | jq -r '.architectures.<architecture>.artifacts.metal.formats.iso.disk.location')
```

5. 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.

6. Boot the RHCOS ISO image without specifying any options, or interrupting the live boot sequence. Wait for the installer to boot into a shell prompt in the RHCOS live environment.



NOTE

You can interrupt the RHCOS installation boot process to add kernel arguments. However, for this ISO procedure you must use the **coreos-installer** command as outlined in the following steps, instead of adding kernel arguments.

7. Run the **coreos-installer** command and specify the options that meet your installation requirements. At a minimum, you must specify the URL that points to the Ignition config file for the node type, and the device that you are installing to:

```
$ sudo coreos-installer install --ignition-url=http://<HTTP_server>/<node_type>.ign <device>
--ignition-hash=sha512-<digest> 1 2
```

- 1 You must run the **coreos-installer** command by using **sudo**, because the **core** user does not have the required root privileges to perform the installation.
- 2 The **--ignition-hash** option is required when the Ignition config file is obtained through an HTTP URL to validate the authenticity of the Ignition config file on the cluster node. **<digest>** is the Ignition config file SHA512 digest obtained in a preceding step.



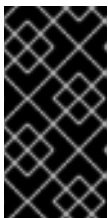
NOTE

If you want to provide your Ignition config files through an HTTPS server that uses TLS, you can add the internal certificate authority (CA) to the system trust store before running **coreos-installer**.

The following example initializes a compute node installation to the **/dev/sda** device. The Ignition config file for the compute node is obtained from an HTTP web server with the IP address 192.168.1.2:

```
$ sudo coreos-installer install --ignition-
url=http://192.168.1.2:80/installation_directory/worker.ign /dev/sda --ignition-hash=sha512-
a5a2d43879223273c9b60af66b44202a1d1248fc01cf156c46d4a79f552b6bad47bc8cc78ddf011
6e80c59d2ea9e32ba53bc807afbca581aa059311def2c3e3b
```

8. Monitor the progress of the RHCOS installation on the console of the machine.



IMPORTANT

Ensure that the installation is successful on each node before commencing with the OpenShift Container Platform installation. Observing the installation process can also help to determine the cause of RHCOS installation issues that might arise.

9. Continue to create more compute machines for your cluster.

3.9.3. Creating 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.



NOTE

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 (**x86_64** + **ppc64le**):

```
kernel http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> initrd=main
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign 1 2
initrd --name main http://<HTTP_server>/rhcos-<version>-live-initramfs.
<architecture>.img 3
boot
```

- 1 Specify the 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.live.rootfs_url** parameter value is the location of the **rootfs** file, and the **coreos.inst.ignition_url** parameter value is the location of the worker Ignition config file.
- 2 If you use multiple NICs, specify a single interface in the **ip** option. For example, to use DHCP on a NIC that is named **eno1**, set **ip=eno1:dhcp**.
- 3 Specify the location of the **initramfs** file that you uploaded to your HTTP server.



NOTE

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?](#) and "Enabling the serial console for PXE and ISO installation" in the "Advanced RHCOS installation configuration" section.



NOTE

To network boot the CoreOS **kernel** on **ppc64le** architecture, you need to use a version of iPXE build with the **IMAGE_GZIP** option enabled. See [IMAGE_GZIP option in iPXE](#).

- For PXE (with UEFI and GRUB as second stage) on **ppc64le**:

```
menuentry 'Install CoreOS' {
  linux rhcos-<version>-live-kernel-<architecture>
  coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
  <architecture>.img coreos.inst.install_dev=/dev/sda
  coreos.inst.ignition_url=http://<HTTP_server>/worker.ign 1 2
  initrd rhcos-<version>-live-initramfs.<architecture>.img 3
}
```

- 1 Specify the locations of the RHCOS files that you uploaded to your HTTP/TFTP server. The **kernel** parameter value is the location of the **kernel** file on your TFTP server. The **coreos.live.rootfs_url** parameter value is the location of the **rootfs** file, and the **coreos.inst.ignition_url** parameter value is the location of the worker Ignition

config file on your HTTP Server.

- 2 If you use multiple NICs, specify a single interface in the **ip** option. For example, to use DHCP on a NIC that is named **eno1**, set **ip=eno1:dhcp**.
- 3 Specify the location of the **initramfs** file that you uploaded to your TFTP server.

2. Use the PXE or iPXE infrastructure to create the required compute machines for your cluster.

3.9.4. 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	v1.33.4
master-1	Ready	master	63m	v1.33.4
master-2	Ready	master	64m	v1.33.4

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
```

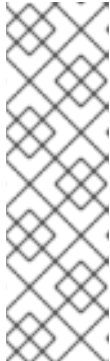
Example output

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.

4. Now that your client requests are approved, you must review the server requests for each machine that you added to the cluster:

```
$ oc get csr
```

Example output

```

NAME      AGE    REQUESTOR                                CONDITION
csr-bfd72 5m26s  system:node:ip-10-0-50-126.us-east-2.compute.internal
Pending
csr-c57lv 5m26s  system:node:ip-10-0-95-157.us-east-2.compute.internal
Pending
...

```

5. If the remaining CSRs are not approved, and are in the **Pending** status, approve the CSRs for your cluster machines:

- To approve them individually, run the following command for each valid CSR:

```
$ oc adm certificate approve <csr_name> 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 -o wide
```

Example output

```

NAME          STATUS ROLES          AGE  VERSION  INTERNAL-IP  EXTERNAL-IP  OS-IMAGE          KERNEL-VERSION  CONTAINER-RUNTIME
worker-0-ppc64le Ready  worker          42d  v1.33.4  192.168.200.21 <none>
Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)  5.14.0-
284.34.1.el9_2.ppc64le cri-o://1.33.4-3.rhaos4.15.gitb36169e.el9
worker-1-ppc64le Ready  worker          42d  v1.33.4  192.168.200.20 <none>
Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)  5.14.0-
284.34.1.el9_2.ppc64le cri-o://1.33.4-3.rhaos4.15.gitb36169e.el9
master-0-x86 Ready  control-plane,master 75d  v1.33.4  10.248.0.38    10.248.0.38
Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)  5.14.0-
284.34.1.el9_2.x86_64 cri-o://1.33.4-3.rhaos4.15.gitb36169e.el9
master-1-x86 Ready  control-plane,master 75d  v1.33.4  10.248.0.39    10.248.0.39
Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)  5.14.0-
284.34.1.el9_2.x86_64 cri-o://1.33.4-3.rhaos4.15.gitb36169e.el9
master-2-x86 Ready  control-plane,master 75d  v1.33.4  10.248.0.40    10.248.0.40
Red Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)  5.14.0-
284.34.1.el9_2.x86_64 cri-o://1.33.4-3.rhaos4.15.gitb36169e.el9
worker-0-x86 Ready  worker          75d  v1.33.4  10.248.0.43    10.248.0.43 Red
Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow)  5.14.0-
284.34.1.el9_2.x86_64 cri-o://1.33.4-3.rhaos4.15.gitb36169e.el9

```

```
worker-1-x86   Ready   worker           75d v1.33.4 10.248.0.44 10.248.0.44 Red
Hat Enterprise Linux CoreOS 415.92.202309261919-0 (Plow) 5.14.0-
284.34.1.el9_2.x86_64 cri-o://1.33.4-3.rhaos4.15.gitb36169e.el9
```

**NOTE**

It can take a few minutes after approval of the server CSRs for the machines to transition to the **Ready** status.

Additional information

- [Certificate Signing Requests](#)

3.10. MANAGING A CLUSTER WITH MULTI-ARCHITECTURE COMPUTE MACHINES

Managing a cluster that has nodes with multiple architectures requires you to consider node architecture as you monitor the cluster and manage your workloads. This requires you to take additional considerations into account when you configure cluster resource requirements and behavior, or schedule workloads in a multi-architecture cluster.

3.10.1. Scheduling workloads on clusters with multi-architecture compute machines

When you deploy workloads on a cluster with compute nodes that use different architectures, you must align pod architecture with the architecture of the underlying node. Your workload may also require additional configuration to particular resources depending on the underlying node architecture.

You can use the Multiarch Tuning Operator to enable architecture-aware scheduling of workloads on clusters with multi-architecture compute machines. The Multiarch Tuning Operator implements additional scheduler predicates in the pods specifications based on the architectures that the pods can support at creation time.

3.10.1.1. Sample multi-architecture node workload deployments

Scheduling a workload to an appropriate node based on architecture works in the same way as scheduling based on any other node characteristic. Consider the following options when determining how to schedule your workloads.

Using **nodeAffinity** to schedule nodes with specific architectures

You can allow a workload to be scheduled on only a set of nodes with architectures supported by its images, you can set the **spec.affinity.nodeAffinity** field in your pod's template specification.

Example deployment with node affinity set

```
apiVersion: apps/v1
kind: Deployment
metadata: # ...
spec:
  # ...
  template:
    # ...
    spec:
      affinity:
```



```

nodeAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
    nodeSelectorTerms:
      - matchExpressions:
          - key: kubernetes.io/arch
            operator: In
            values: 1
          - amd64
          - arm64

```

- 1** Specify the supported architectures. Valid values include **amd64,arm64**, or both values.

Tainting each node for a specific architecture

You can taint a node to avoid the node scheduling workloads that are incompatible with its architecture. When your cluster uses a **MachineSet** object, you can add parameters to the **.spec.template.spec.taints** field to avoid workloads being scheduled on nodes with non-supported architectures.

Before you add a taint to a node, you must scale down the **MachineSet** object or remove existing available machines. For more information, see *Modifying a compute machine set*.

Example machine set with taint set

```

apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata: # ...
spec:
  # ...
  template:
    # ...
    spec:
      # ...
      taints:
        - effect: NoSchedule
          key: multiarch.openshift.io/arch
          value: arm64

```

You can also set a taint on a specific node by running the following command:

```
$ oc adm taint nodes <node-name> multiarch.openshift.io/arch=arm64:NoSchedule
```

Creating a default toleration in a namespace

When a node or machine set has a taint, only workloads that tolerate that taint can be scheduled. You can annotate a namespace so all of the workloads get the same default toleration by running the following command:

Example default toleration set on a namespace

```
$ oc annotate namespace my-namespace \
'scheduler.alpha.kubernetes.io/defaultTolerations'='[{"operator": "Exists", "effect": "NoSchedule",
"key": "multiarch.openshift.io/arch"}]'
```

Tolerating architecture taints in workloads

When a node or machine set has a taint, only workloads that tolerate that taint can be scheduled. You can configure your workload with a **toleration** so that it is scheduled on nodes with specific architecture taints.

Example deployment with toleration set

```
apiVersion: apps/v1
kind: Deployment
metadata: # ...
spec:
  # ...
  template:
    # ...
    spec:
      tolerations:
        - key: "multiarch.openshift.io/arch"
          value: "arm64"
          operator: "Equal"
          effect: "NoSchedule"
```

This example deployment can be scheduled on nodes and machine sets that have the **multiarch.openshift.io/arch=arm64** taint specified.

Using node affinity with taints and tolerations

When a scheduler computes the set of nodes to schedule a pod, tolerations can broaden the set while node affinity restricts the set. If you set a taint on nodes that have a specific architecture, you must also add a toleration to workloads that you want to be scheduled there.

Example deployment with node affinity and toleration set

```
apiVersion: apps/v1
kind: Deployment
metadata: # ...
spec:
  # ...
  template:
    # ...
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: kubernetes.io/arch
                    operator: In
                    values:
                      - amd64
                      - arm64
            tolerations:
              - key: "multiarch.openshift.io/arch"
                value: "arm64"
                operator: "Equal"
                effect: "NoSchedule"
```

Additional resources

- [Managing workloads on multi-architecture clusters by using the Multiarch Tuning Operator](#)
- [Controlling pod placement using node taints](#)
- [Controlling pod placement on nodes using node affinity](#)
- [Controlling pod placement using the scheduler](#)
- [Modifying a compute machine set](#)

3.10.2. Enabling 64k pages on the Red Hat Enterprise Linux CoreOS (RHCOS) kernel

You can enable the 64k memory page in the Red Hat Enterprise Linux CoreOS (RHCOS) kernel on the 64-bit ARM compute machines in your cluster. The 64k page size kernel specification can be used for large GPU or high memory workloads. This is done using the Machine Config Operator (MCO) which uses a machine config pool to update the kernel. To enable 64k page sizes, you must dedicate a machine config pool for ARM64 to enable on the kernel.



IMPORTANT

Using 64k pages is exclusive to 64-bit ARM architecture compute nodes or clusters installed on 64-bit ARM machines. If you configure the 64k pages kernel on a machine config pool using 64-bit x86 machines, the machine config pool and MCO will degrade.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You created a cluster with compute nodes of different architecture on one of the supported platforms.

Procedure

1. Label the nodes where you want to run the 64k page size kernel:

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

Example command

```
$ oc label node worker-arm64-01 node-role.kubernetes.io/worker-64k-pages=
```

2. Create a machine config pool that contains the worker role that uses the ARM64 architecture and the **worker-64k-pages** role:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: worker-64k-pages
spec:
  machineConfigSelector:
```

```

matchExpressions:
  - key: machineconfiguration.openshift.io/role
    operator: In
    values:
      - worker
      - worker-64k-pages
nodeSelector:
  matchLabels:
    node-role.kubernetes.io/worker-64k-pages: ""
    kubernetes.io/arch: arm64

```

3. Create a machine config on your compute node to enable **64k-pages** with the **64k-pages** parameter.

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

Example MachineConfig

```

apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: "worker-64k-pages" 1
  name: 99-worker-64kpages
spec:
  kernelType: 64k-pages 2

```

- 1 Specify the value of the **machineconfiguration.openshift.io/role** label in the custom machine config pool. The example MachineConfig uses the **worker-64k-pages** label to enable 64k pages in the **worker-64k-pages** pool.
- 2 Specify your desired kernel type. Valid values are **64k-pages** and **default**



NOTE

The **64k-pages** type is supported on only 64-bit ARM architecture based compute nodes. The **realtime** type is supported on only 64-bit x86 architecture based compute nodes.

Verification

- To view your new **worker-64k-pages** machine config pool, run the following command:

```
$ oc get mcp
```

Example output

```

NAME      CONFIG                                UPDATED  UPDATING
DEGRADED  MACHINECOUNT  READYMACHINECOUNT  UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT  AGE
master    rendered-master-9d55ac9a91127c36314e1efe7d77fbf8    True    False
False     3              3              3              0              361d

```

worker	rendered-worker-e7b61751c4a5b7ff995d64b967c421ff	True	False
False	7	7	7
		0	361d
worker-64k-pages	rendered-worker-64k-pages-e7b61751c4a5b7ff995d64b967c421ff	True	
False	False	2	2
		2	0
			35m

3.10.3. Importing manifest lists in image streams on your multi-architecture compute machines

On an OpenShift Container Platform 4.20 cluster with multi-architecture compute machines, the image streams in the cluster do not import manifest lists automatically. You must manually change the default **importMode** option to the **PreserveOriginal** option in order to import the manifest list.

Prerequisites

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

Procedure

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

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

Verification

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

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

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

Example output of the **dockerImageManifests** object

```
dockerImageManifests:
  - architecture: amd64
    digest:
      sha256:16d4c96c52923a9968fbfa69425ec703aff711f1db822e4e9788bf5d2bee5d77
    manifestSize: 1252
    mediaType: application/vnd.docker.distribution.manifest.v2+json
    os: linux
  - architecture: arm64
    digest:
      sha256:6ec8ad0d897bcd727531f7d0b716931728999492709d19d8b09f0d90d57f626
    manifestSize: 1252
    mediaType: application/vnd.docker.distribution.manifest.v2+json
    os: linux
  - architecture: ppc64le
    digest:
      sha256:65949e3a80349cdc42acd8c5b34cde6ebc3241eae8daaeaa458498fedb359a6a
    manifestSize: 1252
    mediaType: application/vnd.docker.distribution.manifest.v2+json
```

```
os: linux
- architecture: s390x
  digest:
sha256:75f4fa21224b5d5d511bea8f92dfa8e1c00231e5c81ab95e83c3013d245d1719
  manifestSize: 1252
  mediaType: application/vnd.docker.distribution.manifest.v2+json
os: linux
```

3.11. MANAGING WORKLOADS ON MULTI-ARCHITECTURE CLUSTERS BY USING THE MULTIARCH TUNING OPERATOR

The Multiarch Tuning Operator optimizes workload management within multi-architecture clusters and in single-architecture clusters transitioning to multi-architecture environments.

Architecture-aware workload scheduling allows the scheduler to place pods onto nodes that match the architecture of the pod images.

By default, the scheduler does not consider the architecture of a pod's container images when determining the placement of new pods onto nodes.

To enable architecture-aware workload scheduling, you must create the **ClusterPodPlacementConfig** object. When you create the **ClusterPodPlacementConfig** object, the Multiarch Tuning Operator deploys the necessary operands to support architecture-aware workload scheduling. You can also use the **nodeAffinityScoring** plugin in the **ClusterPodPlacementConfig** object to set cluster-wide scores for node architectures. If you enable the **nodeAffinityScoring** plugin, the scheduler first filters nodes with compatible architectures and then places the pod on the node with the highest score.

When a pod is created, the operands perform the following actions:

1. Add the **multiarch.openshift.io/scheduling-gate** scheduling gate that prevents the scheduling of the pod.
2. Compute a scheduling predicate that includes the supported architecture values for the **kubernetes.io/arch** label.
3. Integrate the scheduling predicate as a **nodeAffinity** requirement in the pod specification.
4. Remove the scheduling gate from the pod.

IMPORTANT

Note the following operand behaviors:

- If the **nodeSelector** field is already configured with the **kubernetes.io/arch** label for a workload, the operand does not update the **nodeAffinity** field for that workload.
- If the **nodeSelector** field is not configured with the **kubernetes.io/arch** label for a workload, the operand updates the **nodeAffinity** field for that workload. However, in that **nodeAffinity** field, the operand updates only the node selector terms that are not configured with the **kubernetes.io/arch** label.
- If the **nodeName** field is already set, the Multiarch Tuning Operator does not process the pod.
- If the pod is owned by a DaemonSet, the operand does not update the **nodeAffinity** field.
- If both **nodeSelector** or **nodeAffinity** and **preferredAffinity** fields are set for the **kubernetes.io/arch** label, the operand does not update the **nodeAffinity** field.
- If only **nodeSelector** or **nodeAffinity** field is set for the **kubernetes.io/arch** label and the **nodeAffinityScoring** plugin is disabled, the operand does not update the **nodeAffinity** field.
- If the **nodeAffinity.preferredDuringSchedulingIgnoredDuringExecution** field already contains terms that score nodes based on the **kubernetes.io/arch** label, the operand ignores the configuration in the **nodeAffinityScoring** plugin.

3.11.1. Installing the Multiarch Tuning Operator by using the CLI

You can install the Multiarch Tuning Operator by using the OpenShift CLI (**oc**).

Prerequisites

- You have installed **oc**.
- You have logged in to **oc** as a user with **cluster-admin** privileges.

Procedure

1. Create a new project named **openshift-multiarch-tuning-operator** by running the following command:

```
$ oc create ns openshift-multiarch-tuning-operator
```

2. Create an **OperatorGroup** object:
 - a. Create a YAML file with the configuration for creating an **OperatorGroup** object.

Example YAML configuration for creating an **OperatorGroup** object

```
apiVersion: operators.coreos.com/v1
```

```
kind: OperatorGroup
metadata:
  name: openshift-multiarch-tuning-operator
  namespace: openshift-multiarch-tuning-operator
spec: {}
```

- b. Create the **OperatorGroup** object by running the following command:

```
$ oc create -f <file_name> 1
```

- 1** Replace **<file_name>** with the name of the YAML file that contains the **OperatorGroup** object configuration.

3. Create a **Subscription** object:

- a. Create a YAML file with the configuration for creating a **Subscription** object.

Example YAML configuration for creating a **Subscription** object

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-multiarch-tuning-operator
  namespace: openshift-multiarch-tuning-operator
spec:
  channel: stable
  name: multiarch-tuning-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Automatic
  startingCSV: multiarch-tuning-operator.<version>
```

- b. Create the **Subscription** object by running the following command:

```
$ oc create -f <file_name> 1
```

- 1** Replace **<file_name>** with the name of the YAML file that contains the **Subscription** object configuration.



NOTE

For more details about configuring the **Subscription** object and **OperatorGroup** object, see "Installing from the software catalog by using the CLI".

Verification

1. To verify that the Multiarch Tuning Operator is installed, run the following command:

```
$ oc get csv -n openshift-multiarch-tuning-operator
```

Example output

NAME	DISPLAY	VERSION	REPLACES
PHASE			
multiarch-tuning-operator.<version>	Multiarch Tuning Operator	<version>	multiarch-tuning-operator.1.0.0
	Succeeded		

The installation is successful if the Operator is in **Succeeded** phase.

- Optional: To verify that the **OperatorGroup** object is created, run the following command:

```
$ oc get operatorgroup -n openshift-multiarch-tuning-operator
```

Example output

NAME	AGE
openshift-multiarch-tuning-operator-q8zbb	133m

- Optional: To verify that the **Subscription** object is created, run the following command:

```
$ oc get subscription -n openshift-multiarch-tuning-operator
```

Example output

NAME	PACKAGE	SOURCE	CHANNEL
multiarch-tuning-operator	multiarch-tuning-operator	redhat-operators	stable

Additional resources

- [Installing from the software catalog using the CLI](#)

3.11.2. Installing the Multiarch Tuning Operator by using the web console

You can install the Multiarch Tuning Operator by using the OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OpenShift Container Platform web console.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Ecosystem** → **Software Catalog**.
3. Enter **Multiarch Tuning Operator** in the search field.
4. Click **Multiarch Tuning Operator**.
5. Select the **Multiarch Tuning Operator** version from the **Version** list.
6. Click **Install**

7. Set the following options on the **Operator Installation** page:

- a. Set **Update Channel** to **stable**.
- b. Set **Installation Mode** to **All namespaces on the cluster**.
- c. Set **Installed Namespace** to **Operator recommended Namespace** or **Select a Namespace**.

The recommended Operator namespace is **openshift-multiarch-tuning-operator**. If the **openshift-multiarch-tuning-operator** namespace does not exist, it is created during the operator installation.

If you select **Select a namespace**, you must select a namespace for the Operator from the **Select Project** list.

- d. **Update approval** as **Automatic** or **Manual**.

If you select **Automatic** updates, Operator Lifecycle Manager (OLM) automatically updates the running instance of the Multiarch Tuning Operator without any intervention.

If you select **Manual** updates, OLM creates an update request. As a cluster administrator, you must manually approve the update request to update the Multiarch Tuning Operator to a newer version.

8. Optional: Select the **Enable Operator recommended cluster monitoring on this Namespace** checkbox.

9. Click **Install**.

Verification

1. Navigate to **Ecosystem** → **Installed Operators**.
2. Verify that the **Multiarch Tuning Operator** is listed with the **Status** field as **Succeeded** in the **openshift-multiarch-tuning-operator** namespace.

3.11.3. Multiarch Tuning Operator pod labels and architecture support overview

After installing the Multiarch Tuning Operator, you can verify the multi-architecture support for workloads in your cluster. You can identify and manage pods based on their architecture compatibility by using the pod labels. These labels are automatically set on the newly created pods to provide insights into their architecture support.

The following table describes the labels that the Multiarch Tuning Operator adds when you create a pod:

Table 3.2. Pod labels that the Multiarch Tuning Operator adds when you create a pod

Label	Description
multiarch.openshift.io/multi-arch: ""	The pod supports multiple architectures.
multiarch.openshift.io/single-arch: ""	The pod supports only a single architecture.
multiarch.openshift.io/arm64: ""	The pod supports the arm64 architecture.

Label	Description
multiarch.openshift.io/amd64: ""	The pod supports the amd64 architecture.
multiarch.openshift.io/ppc64le: ""	The pod supports the ppc64le architecture.
multiarch.openshift.io/s390x: ""	The pod supports the s390x architecture.
multirach.openshift.io/node-affinity: set	The Operator has set the node affinity requirement for the architecture.
multirach.openshift.io/node-affinity: not-set	The Operator did not set the node affinity requirement. For example, when the pod already has a node affinity for the architecture, the Multiarch Tuning Operator adds this label to the pod.
multiarch.openshift.io/scheduling-gate: gated	The pod is gated.
multiarch.openshift.io/scheduling-gate: removed	The pod gate has been removed.
multiarch.openshift.io/inspection-error: ""	An error has occurred while building the node affinity requirements.
multiarch.openshift.io/preferred-node-affinity: set	The Operator has set the architecture preferences in the pod.
multiarch.openshift.io/preferred-node-affinity: not-set	The Operator did not set the architecture preferences in the pod because the user had already set them in the preferredDuringSchedulingIgnoredDuringExecution node affinity.

3.11.4. Creating the ClusterPodPlacementConfig object

After installing the Multiarch Tuning Operator, you must create a **ClusterPodPlacementConfig** object. This object instructs the operator to deploy its operand, which enables architecture-aware workload scheduling across your cluster.

The **ClusterPodPlacementConfig** object supports two optional plugins:

- The **node affinity scoring** plugin patches pods to set soft preferences, using weighted affinities, for the architectures specified by the user. Pods are more likely to be scheduled on nodes running architectures with higher weights.
- The **exec format error monitor** plugin detects **ENOEXEC** errors, which occur when a pod attempts to execute a binary incompatible with the node's architecture. When enabled, this plugin generates events in the affected pod's event stream. It triggers a **ExecFormatErrorsDetected** Prometheus alert if one or more **ENOEXEC** errors are detected

within the last six hours. These errors can result from incorrect architecture node selectors, invalid image metadata that affects architecture-aware workload scheduling, an incorrect binary in an image, or an incompatible binary injected at runtime.



NOTE

You can create only one instance of the **ClusterPodPlacementConfig** object.

Example ClusterPodPlacementConfig object configuration

```
apiVersion: multiarch.openshift.io/v1beta1
kind: ClusterPodPlacementConfig
metadata:
  name: cluster 1
spec:
  logVerbosityLevel: Normal 2
  namespaceSelector: 3
    matchExpressions:
      - key: multiarch.openshift.io/exclude-pod-placement
        operator: DoesNotExist
  plugins: 4
    nodeAffinityScoring: 5
      enabled: true 6
      platforms: 7
        - architecture: amd64 8
          weight: 100 9
        - architecture: arm64
          weight: 50
  execFormatErrorMonitor:
    enabled: true 10
```

- 1 You must set this field value to **cluster**.
- 2 Optional: You can set the field value to **Normal**, **Debug**, **Trace**, or **TraceAll**. The value is set to **Normal** by default.
- 3 Optional: You can configure the **namespaceSelector** to select the namespaces in which the Multiarch Tuning Operator's pod placement operand must process the **nodeAffinity** of the pods. All namespaces are considered by default.
- 4 Optional: Includes a list of plugins for architecture-aware workload scheduling.
- 5 Optional: You can use this plugin to set architecture preferences for pod placement. When enabled, the scheduler first filters out nodes that do not meet the pod's requirements. Then, it prioritizes the remaining nodes based on the architecture scores defined in the **nodeAffinityScoring.platforms** field.
- 6 Optional: Set this field to **true** to enable the **nodeAffinityScoring** plugin. The default value is **false**.
- 7 Optional: Defines a list of architectures and their corresponding scores.
- 8 Specify the node architecture to score. The scheduler prioritizes nodes for pod placement based on the architecture scores that you set and the scheduling requirements defined in the pod

on the architecture scores that you set and the scheduling requirements defined in the pod specification. Accepted values are **arm64**, **amd64**, **ppc64le**, or **s390x**.

- 9 Assign a score to the architecture. The value for this field must be configured in the range of **1** (lowest priority) to **100** (highest priority). The scheduler uses this score to prioritize nodes for pod placement, favoring nodes with architectures that have higher scores.
- 10 Optional: Set this field to **true** to enable the **execFormatErrorMonitor** plugin. When enabled, the plugin detects **ENOEXEC** errors, caused when a pod executes a binary incompatible with the node's architecture. The plugin generates events in the affected pods, and triggers the **ExecFormatErrorsDetected** Prometheus alert if one or more errors are found in the last six hours.

In this example, the **operator** field value is set to **DoesNotExist**. Therefore, if the **key** field value (**multiarch.openshift.io/exclude-pod-placement**) is set as a label in a namespace, the operand does not process the **nodeAffinity** of the pods in that namespace. Instead, the operand processes the **nodeAffinity** of the pods in namespaces that do not contain the label.

If you want the operand to process the **nodeAffinity** of the pods only in specific namespaces, you can configure the **namespaceSelector** as follows:

```
namespaceSelector:
  matchExpressions:
    - key: multiarch.openshift.io/include-pod-placement
      operator: Exists
```

In this example, the **operator** field value is set to **Exists**. Therefore, the operand processes the **nodeAffinity** of the pods only in namespaces that contain the **multiarch.openshift.io/include-pod-placement** label.



IMPORTANT

This Operator excludes pods in namespaces starting with **kube-**. It also excludes pods that are expected to be scheduled on control plane nodes.

3.11.4.1. Creating the ClusterPodPlacementConfig object by using the CLI

To deploy the pod placement operand that enables architecture-aware workload scheduling, you can create the **ClusterPodPlacementConfig** object by using the OpenShift CLI (**oc**).

Prerequisites

- You have installed **oc**.
- You have logged in to **oc** as a user with **cluster-admin** privileges.
- You have installed the Multiarch Tuning Operator.

Procedure

1. Create a **ClusterPodPlacementConfig** object YAML file:

Example ClusterPodPlacementConfig object configuration

```

apiVersion: multiarch.openshift.io/v1beta1
kind: ClusterPodPlacementConfig
metadata:
  name: cluster
spec:
  logVerbosityLevel: Normal
  namespaceSelector:
    matchExpressions:
      - key: multiarch.openshift.io/exclude-pod-placement
        operator: DoesNotExist
  plugins:
    nodeAffinityScoring:
      enabled: true
    platforms:
      - architecture: amd64
        weight: 100
      - architecture: arm64
        weight: 50

```

2. Create the **ClusterPodPlacementConfig** object by running the following command:

```
$ oc create -f <file_name> 1
```

- 1 Replace **<file_name>** with the name of the **ClusterPodPlacementConfig** object YAML file.

Verification

- To check that the **ClusterPodPlacementConfig** object is created, run the following command:

```
$ oc get clusterpodplacementconfig
```

Example output

```

NAME    AGE
cluster 29s

```

3.11.4.2. Creating the ClusterPodPlacementConfig object by using the web console

To deploy the pod placement operand that enables architecture-aware workload scheduling, you can create the **ClusterPodPlacementConfig** object by using the OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OpenShift Container Platform web console.
- You have installed the Multiarch Tuning Operator.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Ecosystem** → **Installed Operators**.
3. On the **Installed Operators** page, click **Multiarch Tuning Operator**.
4. Click the **Cluster Pod Placement Config** tab.
5. Select either **Form view** or **YAML view**.
6. Configure the **ClusterPodPlacementConfig** object parameters.
7. Click **Create**.
8. Optional: If you want to edit the **ClusterPodPlacementConfig** object, perform the following actions:
 - a. Click the **Cluster Pod Placement Config** tab.
 - b. Select **Edit ClusterPodPlacementConfig** from the options menu.
 - c. Click **YAML** and edit the **ClusterPodPlacementConfig** object parameters.
 - d. Click **Save**.

Verification

- On the **Cluster Pod Placement Config** page, check that the **ClusterPodPlacementConfig** object is in the **Ready** state.

3.11.5. Deleting the ClusterPodPlacementConfig object by using the CLI

You can create only one instance of the **ClusterPodPlacementConfig** object. If you want to re-create this object, you must first delete the existing instance.

You can delete this object by using the OpenShift CLI (**oc**).

Prerequisites

- You have installed **oc**.
- You have logged in to **oc** as a user with **cluster-admin** privileges.

Procedure

1. Log in to the OpenShift CLI (**oc**).
2. Delete the **ClusterPodPlacementConfig** object by running the following command:

```
$ oc delete clusterpodplacementconfig cluster
```

Verification

- To check that the **ClusterPodPlacementConfig** object is deleted, run the following command:

```
■
```

```
$ oc get clusterpodplacementconfig
```

Example output

```
No resources found
```

3.11.6. Deleting the ClusterPodPlacementConfig object by using the web console

You can create only one instance of the **ClusterPodPlacementConfig** object. If you want to re-create this object, you must first delete the existing instance.

You can delete this object by using the OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OpenShift Container Platform web console.
- You have created the **ClusterPodPlacementConfig** object.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Ecosystem** → **Installed Operators**.
3. On the **Installed Operators** page, click **Multiarch Tuning Operator**.
4. Click the **Cluster Pod Placement Config** tab.
5. Select **Delete ClusterPodPlacementConfig** from the options menu.
6. Click **Delete**.

Verification

- On the **Cluster Pod Placement Config** page, check that the **ClusterPodPlacementConfig** object has been deleted.

3.11.7. Uninstalling the Multiarch Tuning Operator by using the CLI

You can uninstall the Multiarch Tuning Operator by using the OpenShift CLI (**oc**).

Prerequisites

- You have installed **oc**.
- You have logged in to **oc** as a user with **cluster-admin** privileges.
- You deleted the **ClusterPodPlacementConfig** object.



IMPORTANT

You must delete the **ClusterPodPlacementConfig** object before uninstalling the Multiarch Tuning Operator. Uninstalling the Operator without deleting the **ClusterPodPlacementConfig** object can lead to unexpected behavior.

Procedure

1. Get the **Subscription** object name for the Multiarch Tuning Operator by running the following command:

```
$ oc get subscription.operators.coreos.com -n <namespace> 1
```

- 1 Replace **<namespace>** with the name of the namespace where you want to uninstall the Multiarch Tuning Operator.

Example output

```
NAME                                PACKAGE                SOURCE                CHANNEL
openshift-multiarch-tuning-operator multiarch-tuning-operator redhat-operators stable
```

2. Get the **currentCSV** value for the Multiarch Tuning Operator by running the following command:

```
$ oc get subscription.operators.coreos.com <subscription_name> -n <namespace> -o yaml |
grep currentCSV 1
```

- 1 Replace **<subscription_name>** with the **Subscription** object name. For example: **openshift-multiarch-tuning-operator**. Replace **<namespace>** with the name of the namespace where you want to uninstall the Multiarch Tuning Operator.

Example output

```
currentCSV: multiarch-tuning-operator.<version>
```

3. Delete the **Subscription** object by running the following command:

```
$ oc delete subscription.operators.coreos.com <subscription_name> -n <namespace> 1
```

- 1 Replace **<subscription_name>** with the **Subscription** object name. Replace **<namespace>** with the name of the namespace where you want to uninstall the Multiarch Tuning Operator.

Example output

```
subscription.operators.coreos.com "openshift-multiarch-tuning-operator" deleted
```

4. Delete the CSV for the Multiarch Tuning Operator in the target namespace using the **currentCSV** value by running the following command:

■

```
$ oc delete clusterserviceversion <currentCSV_value> -n <namespace> 1
```

- 1 Replace **<currentCSV>** with the **currentCSV** value for the Multiarch Tuning Operator. For example: **multiarch-tuning-operator.<version>**. Replace **<namespace>** with the name of the namespace where you want to uninstall the Multiarch Tuning Operator.

Example output

```
clusterserviceversion.operators.coreos.com "multiarch-tuning-operator.<version>" deleted
```

Verification

- To verify that the Multiarch Tuning Operator is uninstalled, run the following command:

```
$ oc get csv -n <namespace> 1
```

- 1 Replace **<namespace>** with the name of the namespace where you have uninstalled the Multiarch Tuning Operator.

Example output

```
No resources found in openshift-multiarch-tuning-operator namespace.
```

3.11.8. Uninstalling the Multiarch Tuning Operator by using the web console

You can uninstall the Multiarch Tuning Operator by using the OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster with **cluster-admin** permissions.
- You deleted the **ClusterPodPlacementConfig** object.



IMPORTANT

You must delete the **ClusterPodPlacementConfig** object before uninstalling the Multiarch Tuning Operator. Uninstalling the Operator without deleting the **ClusterPodPlacementConfig** object can lead to unexpected behavior.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Ecosystem → Software Catalog**.
3. Enter **Multiarch Tuning Operator** in the search field.
4. Click **Multiarch Tuning Operator**.
5. Click the **Details** tab.

6. From the **Actions** menu, select **Uninstall Operator**.
7. When prompted, click **Uninstall**.

Verification

1. Navigate to **Ecosystem** → **Installed Operators**.
2. On the **Installed Operators** page, verify that the **Multiarch Tuning Operator** is not listed.

3.12. MULTIARCH TUNING OPERATOR RELEASE NOTES

The Multiarch Tuning Operator (MTO) optimizes workload management within multi-architecture clusters and in single-architecture clusters transitioning to multi-architecture environments.

These release notes track the development of the Multiarch Tuning Operator.

For more information, see [Managing workloads on multi-architecture clusters by using the Multiarch Tuning Operator](#).

3.12.1. Release notes for the Multiarch Tuning Operator 1.2.1

Issued: 15 December 2025

3.12.1.1. Bug fixes

- Previously, the Multiarch Tuning Operator image inspector incorrectly processed images whose registry address included a digest, tag, and port number. The port portion of the registry was incorrectly interpreted as an image tag and was trimmed, causing the inspector to construct an invalid image reference. With this update, image references that contain a digest, tag, and registry port are now correctly parsed and handled. ([MULTIARCH-5767](#))

3.12.2. Release notes for the Multiarch Tuning Operator 1.2.0

Issued: 22 October 2025

3.12.2.1. New features and enhancements

- With this release, you can enable the **exec format error monitor** plugin for the Multiarch Tuning Operator. This plugin detects **ENOEXEC** errors, which occur when a pod attempts to execute a binary incompatible with the node's architecture. You enable this plugin by setting the **plugins.execFormatErrorMonitor.enabled** parameter to **true** in the **ClusterPodPlacementConfig** object. For more information, see [Creating the ClusterPodPlacementConfig object](#).

3.12.2.2. Bug fixes

- Previously, the Multiarch Tuning Operator incorrectly handled the Operator bundle image inspector, restricting it to a single architecture, which could cause OLM to fail when installing Operators. With this update, MTO now sets the bundle image to support all architectures, allowing Operators to be successfully installed on single-architecture clusters when the Multiarch Tuning Operator is deployed. ([MULTIARCH-5546](#))

- Previously, when a cluster global pull secret was changed, stale authentication information could remain in the Multiarch Tuning Operator cache. With this update, the cache is cleared whenever a cluster global pull secret is changed. ([MULTIARCH-5538](#))
- Previously, the Multiarch Tuning Operator failed to process pods if an image reference contained both a tag and a digest. With this update, the image inspector prioritizes the digest if both are present. ([MULTIARCH-5584](#))
- Previously, the Multiarch Tuning Operator did not respect the **.spec.registrySources.containerRuntimeSearchRegistries** field in the **config.openshift.io/Image** custom resource when a workload image did not specify a registry URL. With this update, the Operator can now handle this case, allowing workload images without an explicit registry URL to be pulled successfully. ([MULTIARCH-5611](#))
- Previously, if the **ClusterPodPlacementConfig** object was deleted less than 1 second after its creation, some finalizers were not removed in time, causing certain resources to remain. With this update, all finalizers are properly deleted when the **ClusterPodPlacementConfig** object is deleted. ([MULTIARCH-5372](#))

3.12.3. Release notes for the Multiarch Tuning Operator 1.1.1

Issued: 27 May 2025

3.12.3.1. Bug fixes

- Previously, the pod placement operand did not support authenticating registries using wildcard entries in the hostname of their pull secret. This caused inconsistent behavior with Kubelet when pulling images, because Kubelet supported wildcard entries while the operand required exact hostname matches. As a result, image pulls could fail unexpectedly when registries used wildcard hostnames.
With this release, the pod placement operand supports pull secrets that include wildcard hostnames, ensuring consistent and reliable image authentication and pulling.
- Previously, when image inspection failed after all retries and the **nodeAffinityScoring** plugin was enabled, the pod placement operand applied incorrect **nodeAffinityScoring** labels. With this release, the operand sets **nodeAffinityScoring** labels correctly, even when image inspection fails. It now applies these labels independently of the required affinity process to ensure accurate and consistent scheduling.

3.12.4. Release notes for the Multiarch Tuning Operator 1.1.0

Issued: 18 March 2024

3.12.4.1. New features and enhancements

- The Multiarch Tuning Operator is now supported on managed offerings, including ROSA with Hosted Control Planes (HCP) and other HCP environments.
- With this release, you can configure architecture-aware workload scheduling by using the new **plugins** field in the **ClusterPodPlacementConfig** object. You can use the **plugins.nodeAffinityScoring** field to set architecture preferences for pod placement. If you enable the **nodeAffinityScoring** plugin, the scheduler first filters out nodes that do not meet the pod requirements. Then, the scheduler prioritizes the remaining nodes based on the architecture scores defined in the **nodeAffinityScoring.platforms** field.

3.12.4.1.1. Bug fixes

- With this release, the Multiarch Tuning Operator does not update the **nodeAffinity** field for pods that are managed by a daemon set. ([OCPBUGS-45885](#))

3.12.5. Release notes for the Multiarch Tuning Operator 1.0.0

Issued: 31 October 2024

3.12.5.1. New features and enhancements

- With this release, the Multiarch Tuning Operator supports custom network scenarios and cluster-wide custom registries configurations.
- With this release, you can identify pods based on their architecture compatibility by using the pod labels that the Multiarch Tuning Operator adds to newly created pods.
- With this release, you can monitor the behavior of the Multiarch Tuning Operator by using the metrics and alerts that are registered in the Cluster Monitoring Operator.

CHAPTER 4. POSTINSTALLATION 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.

Resource name	Description
oauth.config.openshift.io	Configures identity providers and other behavior related to internal OAuth server flows.
project.config.openshift.io	Configures how projects are created including the project template.
proxy.config.openshift.io	Defines proxies to be used by components needing external network access. Note: not all components currently consume this value.
scheduler.config.openshift.io	Configures scheduler behavior such as profiles 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 OpenShift image registry settings such as public routing, log levels, proxy settings, resource constraints, replica counts, and storage type.
config.samples.operator.openshift.io	Configures the Samples Operator to control which example image streams and templates are installed on the cluster.

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 4.20, 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 check 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. ADDING WORKER NODES

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

4.2.1. Adding worker nodes to an on-premise cluster

For on-premise clusters, you can add worker nodes by using the OpenShift Container Platform CLI (**oc**) to generate an ISO image, which can then be used to boot one or more nodes in your target cluster. This process can be used regardless of how you installed your cluster.

You can add one or more nodes at a time while customizing each node with more complex configurations, such as static network configuration, or you can specify only the MAC address of each node. Any configurations that are not specified during ISO generation are retrieved from the target cluster and applied to the new nodes.

Preflight validation checks are also performed when booting the ISO image to inform you of failure-causing issues before you attempt to boot each node.

[Adding worker nodes to an on-premise cluster](#)

4.2.2. Adding worker nodes to installer-provisioned infrastructure clusters

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

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

- [Adding worker nodes using the web console](#)
- [Adding worker nodes using YAML in the web console](#)
- [Manually adding a worker node to an installer-provisioned infrastructure cluster](#)

4.2.3. Adding worker nodes to user-provisioned infrastructure clusters

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

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

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

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

- [Adding worker nodes using the OpenShift Cluster Manager](#)
- [Adding worker nodes using the Assisted Installer REST API](#)
- [Manually adding worker nodes to a SNO cluster](#)

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

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

- [Creating your cluster with the console](#)

4.3. ADJUST WORKER NODES

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

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

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

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

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

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

4.3.2. Scaling a compute machine set manually

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

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

Prerequisites

- Install an OpenShift Container Platform cluster and the **oc** command line.
- Log in to **oc** as a user with **cluster-admin** permission.

Procedure

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

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

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

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

```
$ oc get machines.machine.openshift.io -n openshift-machine-api
```

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

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

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

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

Or:

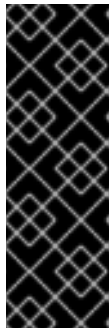
```
$ oc edit machinesets.machine.openshift.io <machineset> -n openshift-machine-api
```

TIP

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

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

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

**IMPORTANT**

By default, the machine controller tries to drain the node that is backed by the machine until it succeeds. In some situations, such as with a misconfigured pod disruption budget, the drain operation might not be able to succeed. If the drain operation fails, the machine controller cannot proceed removing the machine.

You can skip draining the node by annotating **machine.openshift.io/exclude-node-draining** in a specific machine.

Verification

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

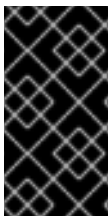
```
$ oc get machines.machine.openshift.io
```

4.3.3. The compute machine set deletion policy

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

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

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

**IMPORTANT**

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

**NOTE**

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

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 compute machine set, or a machine config. Adding the label to the compute machine set ensures that if the node or machine goes down, new nodes have the label. Labels added to a node or machine config do not persist if the node or machine goes down.

**NOTE**

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

Procedure

To add a default cluster-wide node selector:

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

```
$ oc edit scheduler cluster
```

Example Scheduler Operator CR with a node selector

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
...
spec:
  defaultNodeSelector: type=user-node,region=east 1
  mastersSchedulable: false
```

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

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

2. Add labels to a node by using a compute machine set or editing the node directly:
 - Use a compute machine set to add labels to nodes managed by the compute machine set when a node is created:

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

```
$ oc patch MachineSet <name> --type='json' -
p=[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"<key>="
<value>","<key>="<value>"}]}] -n openshift-machine-api 1
```

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

For example:

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

TIP

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

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

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

For example:

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

Example MachineSet object

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
...
spec:
...
template:
  metadata:
...
spec:
  metadata:
    labels:
```

```

    region: east
    type: user-node
  ...

```

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

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

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

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

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

For example:

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

Example output

```

NAME                                STATUS ROLES  AGE  VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-c-vmqzp Ready  worker  61s  v1.33.4

```

- 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  v1.33.4
```

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

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

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

1. The node controller on the control plane updates the node health to **Unhealthy** and marks the node **Ready** condition `Unknown``.
2. In response, the scheduler stops scheduling pods to that node.
3. The Node Lifecycle Controller adds a **node.kubernetes.io/unreachable** taint with a **NoExecute** effect to the node and schedules any pods on the node for eviction after five minutes, by default.

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

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

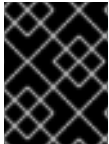
These worker latency profiles contain three sets of parameters that are predefined with carefully tuned values to control the reaction of the cluster to increased latency. There is no need to experimentally find the best values manually.

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

4.4.1. Understanding worker latency profiles

Review the following information to learn about worker latency profiles, which allow you to control the reaction of the cluster to latency issues without needing to determine the best values by using manual methods.

Worker latency profiles are four different categories of carefully-tuned parameters. The four parameters which implement these values are **node-status-update-frequency**, **node-monitor-grace-period**, **default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds**.



IMPORTANT

Setting these parameters manually is not supported. Incorrect parameter settings adversely affect cluster stability.

All worker latency profiles configure the following parameters:

node-status-update-frequency

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

node-monitor-grace-period

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

default-not-ready-toleration-seconds

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

default-unreachable-toleration-seconds

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

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

- The Machine Config Operator (MCO) updates the **node-status-update-frequency** parameter on the worker nodes.
- The Kubernetes Controller Manager updates the **node-monitor-grace-period** parameter on the control plane nodes.
- The Kubernetes API Server Operator updates the **default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds** parameters on the control plane nodes.

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

Default worker latency profile

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

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

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

Medium worker latency profile

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

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

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

Low worker latency profile

Use the **LowUpdateSlowReaction** profile if the network latency is extremely high.

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

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

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



NOTE

The latency profiles do not support custom machine config pools, only the default worker machine config pools.

4.4.2. Using and changing worker latency profiles

You can change a worker latency profile to deal with network latency at any time by editing the **node.config** object. This allows you to ensure that your cluster runs properly if network latency between the control plane and the worker nodes fluctuates.

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



NOTE

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

Procedure

1. Move to the medium worker latency profile:

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

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

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

Example node.config object

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

# ...
```

where:

spec.workerLatencyProfile.MediumUpdateAverageReaction

Specifies that the medium worker latency policy should be used.

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

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

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

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

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

Example node.config object

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
  annotations:
```

```

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

```

where:

spec.workerLatencyProfile.LowUpdateSlowReaction

Specifies that the low worker latency policy should be used.

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

Verification

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

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

Example output

```

# ...
- lastTransitionTime: "2022-07-11T19:47:10Z"
  reason: ProfileUpdated
  status: "False"
  type: WorkerLatencyProfileProgressing
- lastTransitionTime: "2022-07-11T19:47:10Z"
  message: all static pod revision(s) have updated latency profile
  reason: ProfileUpdated
  status: "True"
  type: WorkerLatencyProfileComplete
- lastTransitionTime: "2022-07-11T19:20:11Z"
  reason: AsExpected
  status: "False"
  type: WorkerLatencyProfileDegraded
- lastTransitionTime: "2022-07-11T19:20:36Z"
  status: "False"
# ...

```

where:

status.message: all static pod revision(s) have updated latency profile

Specifies that the profile is applied and active.

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

4.5. MANAGING CONTROL PLANE MACHINES

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

4.5.1. Adding a control plane node to your cluster

When installing a cluster on bare-metal infrastructure, you can manually scale up to 4 or 5 control plane nodes for your cluster. The example in the procedure uses **node-5** as the new control plane node.

Prerequisites

- You have installed a healthy cluster with at least three control plane nodes.
- You have created a single control plane node that you intend to add to your cluster as a postinstallation task.

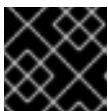
Procedure

1. Retrieve pending Certificate Signing Requests (CSRs) for the new control plane node by entering the following command:

```
$ oc get csr | grep Pending
```

2. Approve all pending CSRs for the control plane node by entering 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
```



IMPORTANT

You must approve the CSRs to complete the installation.

3. Confirm that the control plane node is in the **Ready** status by entering the following command:

```
$ oc get nodes
```



NOTE

On installer-provisioned infrastructure, the etcd Operator relies on the Machine API to manage the control plane and ensure etcd quorum. The Machine API then uses **Machine** CRs to represent and manage the underlying control plane nodes.

4. Create the **BareMetalHost** and **Machine** CRs and link them to the **Node** CR of the control plane node.

- a. Create the **BareMetalHost** CR with a unique **.metadata.name** value as demonstrated in the following example:

```
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: node-5
  namespace: openshift-machine-api
spec:
  automatedCleaningMode: metadata
  bootMACAddress: 00:00:00:00:00:02
  bootMode: UEFI
  customDeploy:
    method: install_coreos
  externallyProvisioned: true
  online: true
  userData:
    name: master-user-data-managed
    namespace: openshift-machine-api
# ...
```

- b. Apply the **BareMetalHost** CR by entering the following command:

```
$ oc apply -f <filename> ❶
```

- ❶ ❶ Replace <filename> with the name of the **BareMetalHost** CR.

- c. Create the **Machine** CR by using the unique **.metadata.name** value as demonstrated in the following example:

```
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  annotations:
    machine.openshift.io/instance-state: externally provisioned
    metal3.io/BareMetalHost: openshift-machine-api/node-5
  finalizers:
    - machine.machine.openshift.io
  labels:
    machine.openshift.io/cluster-api-cluster: <cluster_name> ❶
    machine.openshift.io/cluster-api-machine-role: master
    machine.openshift.io/cluster-api-machine-type: master
  name: node-5
  namespace: openshift-machine-api
spec:
  metadata: {}
  providerSpec:
    value:
      apiVersion: baremetal.cluster.k8s.io/v1alpha1
      customDeploy:
        method: install_coreos
```

```

hostSelector: {}
image:
  checksum: ""
  url: ""
kind: BareMetalMachineProviderSpec
metadata:
  creationTimestamp: null
userData:
  name: master-user-data-managed
# ...

```

- 1 Replace **<cluster_name>** with the name of the specific cluster, for example, **test-day2-1-6qv96**.

- d. Get the cluster name by running the following command:

```
$ oc get infrastructure cluster -o=jsonpath='{.status.infrastructureName}'{"\n"}
```

- e. Apply the **Machine** CR by entering the following command:

```
$ oc apply -f <filename> 1
```

- 1 Replace **<filename>** with the name of the **Machine** CR.

- f. Link **BareMetalHost**, **Machine**, and **Node** objects by running the **link-machine-and-node.sh** script:

- i. Copy the following **link-machine-and-node.sh** script to a local machine:

```

#!/bin/bash

# Credit goes to
# https://bugzilla.redhat.com/show_bug.cgi?id=1801238.
# This script will link Machine object
# and Node object. This is needed
# in order to have IP address of
# the Node present in the status of the Machine.

set -e

machine="$1"
node="$2"

if [ -z "$machine" ] || [ -z "$node" ]; then
    echo "Usage: $0 MACHINE NODE"
    exit 1
fi

node_name=$(echo "${node}" | cut -f2 -d':')

oc proxy &
proxy_pid=$!
function kill_proxy {

```

```

    kill $proxy_pid
}
trap kill_proxy EXIT SIGINT

HOST_PROXY_API_PATH="http://localhost:8001/apis/metal3.io/v1alpha1/namespaces/openshift-machine-api/baremetalhosts"

function print_nics() {
    local ips
    local eob
    declare -a ips

    readarray -t ips <<(echo "${1}" \
        | jq '.[] | select(.type == "InternalIP") | .address' \
        | sed 's/"//g')

    eob=','
    for (( i=0; i<${#ips[@]}; i++ )); do
        if [ ${!(i+1)} -eq ${#ips[@]} ]; then
            eob=""
        fi
        cat <<- EOF
        {
            "ip": "${ips[$i]}",
            "mac": "00:00:00:00:00:00",
            "model": "unknown",
            "speedGbps": 10,
            "vlanId": 0,
            "pxe": true,
            "name": "eth1"
        }${eob}
    EOF
    done
}

function wait_for_json() {
    local name
    local url
    local curl_opts
    local timeout

    local start_time
    local curr_time
    local time_diff

    name="$1"
    url="$2"
    timeout="$3"
    shift 3
    curl_opts="$@"
    echo -n "Waiting for $name to respond"
    start_time=$(date +%s)
    until curl -g -X GET "$url" "${curl_opts[@]}" 2> /dev/null | jq '.' 2> /dev/null > /dev/null; do
        echo -n "."
        curr_time=$(date +%s)

```



```

        time_diff=$((curr_time - start_time))
        if [[ $time_diff -gt $timeout ]]; then
            printf "\nTimed out waiting for %s" "${name}"
            return 1
        fi
        sleep 5
    done
    echo " Success!"
    return 0
}
wait_for_json oc_proxy "${HOST_PROXY_API_PATH}" 10 -H "Accept:
application/json" -H "Content-Type: application/json"

addresses=$(oc get node -n openshift-machine-api "${node_name}" -o json | jq -c
'.status.addresses')

machine_data=$(oc get machines.machine.openshift.io -n openshift-machine-api -o
json "${machine}")
host=$(echo "$machine_data" | jq
'.metadata.annotations["metal3.io/BareMetalHost"]' | cut -f2 -d/ | sed 's//g')

if [ -z "$host" ]; then
    echo "Machine $machine is not linked to a host yet." 1>&2
    exit 1
fi

# The address structure on the host doesn't match the node, so extract
# the values we want into separate variables so we can build the patch
# we need.
hostname=$(echo "${addresses}" | jq '.[] | select(.type == "Hostname") | .address' |
sed 's//g')

set +e
read -r -d " host_patch << EOF
{
    "status": {
        "hardware": {
            "hostname": "${hostname}",
            "nics": [
$(print_nics "${addresses}")
            ],
            "systemVendor": {
                "manufacturer": "Red Hat",
                "productName": "product name",
                "serialNumber": ""
            },
            "firmware": {
                "bios": {
                    "date": "04/01/2014",
                    "vendor": "SeaBIOS",
                    "version": "1.11.0-2.el7"
                }
            },
            "ramMebibytes": 0,
            "storage": [],
            "cpu": {

```

```

        "arch": "x86_64",
        "model": "Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz",
        "clockMegahertz": 2199.998,
        "count": 4,
        "flags": []
      }
    }
  }
}
EOF
set -e

echo "PATCHING HOST"
echo "${host_patch}" | jq .

curl -s \
  -X PATCH \
  "${HOST_PROXY_API_PATH}/${host}/status" \
  -H "Content-type: application/merge-patch+json" \
  -d "${host_patch}"

oc get baremetalhost -n openshift-machine-api -o yaml "${host}"

```

- ii. Make the script executable by entering the following command:

```
$ chmod +x link-machine-and-node.sh
```

- iii. Run the script by entering the following command:

```
$ bash link-machine-and-node.sh node-5 node-5
```



NOTE

The first **node-5** instance represents the machine, and the second instance represents the node.

Verification

1. Confirm members of etcd by executing into one of the pre-existing control plane nodes:
 - a. Open a remote shell session to the control plane node by entering the following command:

```
$ oc rsh -n openshift-etcd etcd-node-0
```

- b. List etcd members:

```
# etcdctl member list -w table
```

2. Check the etcd Operator configuration process until completion by entering the following command. Expected output shows **False** under the **PROGRESSING** column.

```
$ oc get clusteroperator etcd
```

3. Confirm etcd health by running the following commands:

- a. Open a remote shell session to the control plane node:

```
$ oc rsh -n openshift-etcd etcd-node-0
```

- b. Check endpoint health. Expected output shows **is healthy** for the endpoint.

```
# etcdctl endpoint health
```

4. Verify that all nodes are ready by entering the following command. The expected output shows the **Ready** status beside each node entry.

```
$ oc get nodes
```

5. Verify that the cluster Operators are all available by entering the following command. Expected output lists each Operator and shows the available status as **True** beside each listed Operator.

```
$ oc get ClusterOperators
```

6. Verify that the cluster version is correct by entering the following command:

```
$ oc get ClusterVersion
```

Example output

```
NAME      VERSION AVAILABLE PROGRESSING SINCE STATUS
version   OpenShift Container Platform.5 True      False      5h57m Cluster version is
OpenShift Container Platform.5
```

4.6. CREATING INFRASTRUCTURE MACHINE SETS FOR PRODUCTION ENVIRONMENTS

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

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

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

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

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

4.6.1. Creating a compute machine set

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

Prerequisites

- Deploy an OpenShift Container Platform cluster.
- Install the OpenShift CLI (**oc**).
- Log in to **oc** as a user with **cluster-admin** permission.

Procedure

1. Create a new YAML file that contains the compute machine set custom resource (CR) sample and is named **<file_name>.yaml**.
Ensure that you set the **<clusterID>** and **<role>** parameter values.
2. Optional: If you are not sure which value to set for a specific field, you can check an existing compute machine set from your cluster.
 - a. To list the compute machine sets in your cluster, run the following command:

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

Example output

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

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

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

Example output

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id> 1
    name: <infrastructure_id>-<role> 2
    namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
```

```

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

```

1 The cluster infrastructure ID.

2 A default node label.



NOTE

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

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

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

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

Verification

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

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

Example output

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

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

4.6.2. 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 (infra) nodes, be provisioned. The installation program provisions only control plane and worker nodes. Worker nodes can be designated as infrastructure nodes through labeling. You can then use taints and tolerations to move appropriate workloads to the infrastructure nodes. For more information, see "Moving resources to infrastructure machine sets".

You can optionally create a default cluster-wide node selector. The default node selector is applied to pods created in all namespaces and 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.

Procedure

1. 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=""
```

2. Check to see if applicable nodes now have the **infra** role:

```
$ oc get nodes
```

3. Optional: Create a default cluster-wide node selector:

- a. Edit the **Scheduler** object:

```
$ oc edit scheduler cluster
```

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

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

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

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

c. Save the file to apply the changes.

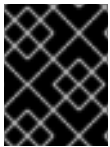
You can now move infrastructure resources to the new infrastructure nodes. Also, remove any workloads that you do not want, or that do not belong, on the new infrastructure node. See the list of workloads supported for use on infrastructure nodes in "OpenShift Container Platform infrastructure components".

Additional resources

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

4.6.3. Creating a machine config pool for infrastructure machines

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



IMPORTANT

Creating a custom machine configuration pool overrides default worker pool configurations if they refer to the same file or unit.

Procedure

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

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

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

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

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

Example output

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

```
nodeSelector:
  matchLabels:
    node-role.kubernetes.io/infra: "" 2
```

- 1 Add the worker role and your custom role.
- 2 Add the label you added to the node as a **nodeSelector**.



NOTE

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

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

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

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

```
$ oc get machineconfig
```

Example output

NAME		GENERATEDBYCONTROLLER	
IGNITIONVERSION	CREATED		
00-master		365c1cfd14de5b0e3b85e0fc815b0060f36ab955	
3.5.0	31d		
00-worker		365c1cfd14de5b0e3b85e0fc815b0060f36ab955	
3.5.0	31d		
01-master-container-runtime			
365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.5.0	31d	
01-master-kubelet		365c1cfd14de5b0e3b85e0fc815b0060f36ab955	
3.5.0	31d		
01-worker-container-runtime			
365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.5.0	31d	
01-worker-kubelet		365c1cfd14de5b0e3b85e0fc815b0060f36ab955	
3.5.0	31d		
99-master-1ae2a1e0-a115-11e9-8f14-005056899d54-registries			
365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.5.0	31d	
99-master-ssh			3.2.0 31d
99-worker-1ae64748-a115-11e9-8f14-005056899d54-registries			
365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.5.0	31d	
99-worker-ssh			3.2.0 31d
rendered-infra-4e48906dca84ee702959c71a53ee80e7			
365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.5.0	23m	
rendered-master-072d4b2da7f88162636902b074e9e28e5b6fb8349a29735e48446d435962dec4547d3090	3.5.0	31d	
rendered-master-3e88ec72aed3886dec061df60d16d1af02c07496ba0417b3e12b78fb32baf6293d314f79	3.5.0	31d	

rendered-master-419bee7de96134963a15fdf9dd473b25365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.5.0	17d
rendered-master-53f5c91c7661708adce18739cc0f40fb365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.5.0	13d
rendered-master-a6a357ec18e5bce7f5ac426fc7c5ffcd365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.5.0	7d3h
rendered-master-dc7f874ec77fc4b969674204332da0375b6fb8349a29735e48446d435962dec4547d3090	3.5.0	31d
rendered-worker-1a75960c52ad18ff5dfa6674eb7e533d5b6fb8349a29735e48446d435962dec4547d3090	3.5.0	31d
rendered-worker-2640531be11ba43c61d72e82dc634ce65b6fb8349a29735e48446d435962dec4547d3090	3.5.0	31d
rendered-worker-4e48906dca84ee702959c71a53ee80e7365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.5.0	7d3h
rendered-worker-4f110718fe88e5f349987854a1147755365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.5.0	17d
rendered-worker-afc758e194d6188677eb837842d3b37902c07496ba0417b3e12b78fb32baf6293d314f79	3.5.0	31d
rendered-worker-daa08cc1e8f5fcdeba24de60cd955cc3365c1cfd14de5b0e3b85e0fc815b0060f36ab955	3.5.0	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: 3.5.0
    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.

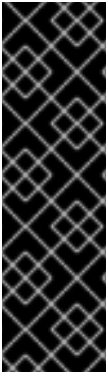
4.7. ASSIGNING MACHINE SET RESOURCES TO INFRASTRUCTURE NODES

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.

4.7.1. Binding infrastructure node workloads using taints and tolerations

If you have an infrastructure 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 infrastructure 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 infrastructure 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=reserved:NoSchedule
...
```

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

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

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

For example:

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

TIP

You can alternatively edit the pod specification to add the taint:

```
apiVersion: v1
kind: Node
metadata:
  name: node1
# ...
spec:
  taints:
    - key: node-role.kubernetes.io/infra
      value: reserved
      effect: NoSchedule
# ...
```

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

If you added a **NoSchedule** taint to the infrastructure node, any pods that are controlled by a daemon set on that node are marked as **misscheduled**. You must either delete the pods or add a toleration to the pods as shown in the Red Hat Knowledgebase solution [add toleration on misscheduled DNS pods](#). Note that you cannot add a toleration to a daemon set object that is managed by an operator.

**NOTE**

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

2. Add tolerations to the pods that you want to schedule on the infrastructure node, such as the router, registry, and monitoring workloads. Referencing the previous examples, add the following tolerations to the **Pod** object specification:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
# ...
spec:
# ...
  tolerations:
    - key: node-role.kubernetes.io/infra ❶
      value: reserved ❷
      effect: NoSchedule ❸
      operator: Equal ❹
```

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

- 3 Specify the effect that you added to the node.
- 4 Specify the **Equal** 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 infrastructure node.



NOTE

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

3. Schedule the pod to the infrastructure node by using a scheduler. See the documentation for "Controlling pod placement using the scheduler" for details.
4. Remove any workloads that you do not want, or that do not belong, on the new infrastructure node. See the list of workloads supported for use on infrastructure nodes in "OpenShift Container Platform infrastructure components".

Additional resources

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

4.8. 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.

4.8.1. Moving the router

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

Prerequisites

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

Procedure

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

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

The command output resembles the following text:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: 2019-04-18T12:35:39Z
finalizers:
```

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

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

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

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: "2025-03-26T21:15:43Z"
  finalizers:
    - ingresscontroller.operator.openshift.io/finalizer-ingresscontroller
  generation: 1
  name: default
# ...
spec:
  nodePlacement:
    nodeSelector: 1
    matchLabels:
      node-role.kubernetes.io/infra: ""
  tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
# ...
```

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

3. Confirm that the router pod is running on the **infra** node.
 - a. View the list of router pods and note the node name of the running pod:

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

Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
NOMINATED NODE READINESS GATES						
router-default-86798b4b5d-bdlvd	1/1	Running	0	28s	10.130.2.4	ip-10-0-217-226.ec2.internal
router-default-955d875f4-255g8	0/1	Terminating	0	19h	10.129.2.4	ip-10-0-148-172.ec2.internal

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	v1.33.4

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

4.8.2. Moving the default registry

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

Prerequisites

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

Procedure

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

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

Example output

```
apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  creationTimestamp: 2019-02-05T13:52:05Z
  finalizers:
  - imageregistry.operator.openshift.io/finalizer
  generation: 1
  name: cluster
  resourceVersion: "56174"
  selfLink: /apis/imageregistry.operator.openshift.io/v1/configs/cluster
  uid: 36fd3724-294d-11e9-a524-12fjee2931b
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
```

```

apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  name: cluster
# ...
spec:
  logLevel: Normal
  managementState: Managed
  nodeSelector: 1
    node-role.kubernetes.io/infra: ""
  tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved

```

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

3. 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.

4.8.3. Moving the monitoring solution

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

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have created the **cluster-monitoring-config ConfigMap** object.
- You have installed the OpenShift CLI (**oc**).

Procedure

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

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

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |+
    alertmanagerMain:
      nodeSelector: 1
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
    prometheusK8s:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
    prometheusOperator:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
    metricsServer:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - key: node-role.kubernetes.io/infra
        value: reserved
        effect: NoSchedule
    kubeStateMetrics:
```

```

nodeSelector:
  node-role.kubernetes.io/infra: ""
tolerations:
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoSchedule
telemetryClient:
nodeSelector:
  node-role.kubernetes.io/infra: ""
tolerations:
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoSchedule
openshiftStateMetrics:
nodeSelector:
  node-role.kubernetes.io/infra: ""
tolerations:
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoSchedule
thanosQuerier:
nodeSelector:
  node-role.kubernetes.io/infra: ""
tolerations:
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoSchedule
monitoringPlugin:
nodeSelector:
  node-role.kubernetes.io/infra: ""
tolerations:
- key: node-role.kubernetes.io/infra
  value: reserved
  effect: NoSchedule

```

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

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

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

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

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

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

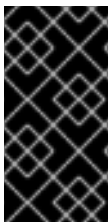
4.9. ABOUT THE CLUSTER AUTOSCALER

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.

4.9.1. Automatic node removal

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

- The node utilization is less than the *node utilization level* threshold for the cluster. The node utilization level is the sum of the requested resources divided by the allocated resources for the node. If you do not specify a value in the **ClusterAutoscaler** custom resource, the cluster autoscaler uses a default value of **0.5**, which corresponds to 50% utilization.
- The cluster autoscaler can move all pods running on the node to the other nodes. The Kubernetes scheduler is responsible for scheduling pods on the nodes.
- The cluster autoscaler does not have scale down disabled annotation.

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

- Pods with restrictive pod disruption budgets (PDBs).
- Kube-system pods that do not run on the node by default.
- Kube-system pods that do not have a PDB or have a PDB that is too restrictive.
- Pods that are not backed by a controller object such as a deployment, replica set, or stateful set.
- Pods with local storage.
- Pods that cannot be moved elsewhere because of a lack of resources, incompatible node selectors or affinity, matching anti-affinity, and so on.
- Unless they also have a **"cluster-autoscaler.kubernetes.io/safe-to-evict": "true"** annotation, pods that have a **"cluster-autoscaler.kubernetes.io/safe-to-evict": "false"** annotation.

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

4.9.2. Limitations

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.



NOTE

The cluster autoscaler only adds nodes in autoscaled node groups if doing so would result in a schedulable pod. If the available node types cannot meet the requirements for a pod request, or if the node groups that could meet these requirements are at their maximum size, the cluster autoscaler cannot scale up.

4.9.3. Interaction with other scheduling features

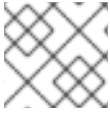
The horizontal pod autoscaler (HPA) and the cluster autoscaler modify cluster resources in different ways. The HPA changes the deployment's or replica set's number of replicas based on the current CPU load. If the load increases, the HPA creates new replicas, regardless of the amount of resources available 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.

4.9.4. Cluster autoscaler resource definition

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

**NOTE**

When you change the configuration of an existing cluster autoscaler, it restarts.

```

apiVersion: "autoscaling.openshift.io/v1"
kind: "ClusterAutoscaler"
metadata:
  name: "default"
spec:
  podPriorityThreshold: -10
  resourceLimits:
    maxNodesTotal: 24
    cores:
      min: 8
      max: 128
    memory:
      min: 4
      max: 256
    gpus:
      - type: <gpu_type>
        min: 0
        max: 16
  logVerbosity: 4
  scaleDown:
    enabled: true
    delayAfterAdd: 10m
    delayAfterDelete: 5m
    delayAfterFailure: 30s
    unneededTime: 5m
    utilizationThreshold: "0.4"
  scaleUp:
    newPodScaleUpDelay: "10s"
  expanders: ["Random"]

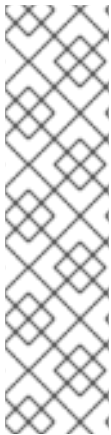
```

Table 4.1. Cluster autoscaler parameters

Parameter	Description
podPriorityThreshold	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.
maxNodesTotal	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.
cores.min	Specify the minimum number of cores to deploy in the cluster.
cores.max	Specify the maximum number of cores to deploy in the cluster.

Parameter	Description
memory.min	Specify the minimum amount of memory, in GiB, in the cluster.
memory.max	Specify the maximum amount of memory, in GiB, in the cluster.
gpus.type	Optional: To configure the cluster autoscaler to deploy GPU-enabled nodes, specify a type value. This value must match the value of the spec.template.spec.metadata.labels[cluster-api/accelerator] label in the machine set that manages the GPU-enabled nodes of that type. For example, this value might be nvidia-t4 to represent Nvidia T4 GPUs, or nvidia-a10g for A10G GPUs. For more information, see "Labeling GPU machine sets for the cluster autoscaler".
gpus.min	Specify the minimum number of GPUs of the specified type to deploy in the cluster.
gpus.max	Specify the maximum number of GPUs of the specified type to deploy in the cluster.
logVerbosity	Specify the logging verbosity level between 0 and 10. The following log level thresholds are provided for guidance: ● 1: (Default) Basic information about changes. ● 4: Debug-level verbosity for troubleshooting typical issues. ● 9: Extensive, protocol-level debugging information. If you do not specify a value, the default value of 1 is used.
scaleDown	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 .
scaleDown.enabled	Specify whether the cluster autoscaler can remove unnecessary nodes.
scaleDown.delayAfterAdd	Optional: Specify the period to wait before deleting a node after a node has recently been <i>added</i> . If you do not specify a value, the default value of 10m is used.
scaleDown.delayAfterDelete	Optional: Specify the period to wait before deleting a node after a node has recently been <i>deleted</i> . If you do not specify a value, the default value of 10s is used.
scaleDown.delayAfterFailure	Optional: Specify the period to wait before deleting a node after a scale down failure occurred. If you do not specify a value, the default value of 3m is used.
scaleDown.unneededTime	Optional: Specify a period of time before an unnecessary node is eligible for deletion. If you do not specify a value, the default value of 10m is used.

Parameter	Description
scaleDown.utilizationThreshold	Optional: Specify the <i>node utilization level</i>. Nodes below this utilization level are eligible for deletion. The node utilization level is the sum of the requested resources divided by the allocated resources for the node, and must be a value greater than "0" but less than "1". If you do not specify a value, the cluster autoscaler uses a default value of "0.5", which corresponds to 50% utilization. You must express this value as a string.
scaleUp	In this section, you can specify the period to wait before recognizing newly pending pods by using any valid ParseDuration interval, including ns, us, ms, s, m, and h.
scaleUp.newPodScaleUpDelay	Optional: Specify the period to ignore a new unschedulable pod before adding a new node. If you do not specify a value, the default value of 0s is used.
expanders	Optional: Specify any expanders that you want the cluster autoscaler to use. The following values are valid: ● LeastWaste: Selects the machine set that minimizes the idle CPU after scaling. If multiple machine sets would yield the same amount of idle CPU, the selection minimizes unused memory. ● Priority: Selects the machine set with the highest user-assigned priority. To use this expander, you must create a config map that defines the priority of your machine sets. For more information, see "Configuring a priority expander for the cluster autoscaler." ● Random: (Default) Selects the machine set randomly. If you do not specify a value, the default value of Random is used. You can specify multiple expanders by using the [LeastWaste, Priority] format. The cluster autoscaler applies each expander according to the specified order. In the [LeastWaste, Priority] example, the cluster autoscaler first evaluates according to the LeastWaste criteria. If more than one machine set satisfies the LeastWaste criteria equally well, the cluster autoscaler then evaluates according to the Priority criteria. If more than one machine set satisfies all of the specified expanders equally well, the cluster autoscaler selects one to use at random.



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.

4.9.5. Deploying a cluster autoscaler

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

Procedure

1. Create a YAML file for a **ClusterAutoscaler** resource that contains the custom resource definition.
2. Create the custom resource in the cluster by running the following command:

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

where:

<filename>

Specifies the name of the YAML file you created.

4.10. APPLYING AUTOSCALING TO YOUR CLUSTER

Applying autoscaling to an OpenShift Container Platform cluster involves deploying a cluster autoscaler and then deploying machine autoscalers for each machine type in your cluster.

For more information, see [Applying autoscaling to an OpenShift Container Platform cluster](#) .

4.11. 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).

4.11.1. Understanding feature gates

You can use the **FeatureGate** custom resource (CR) to enable specific feature sets so that you can use specific non-default features in your cluster.

A feature set is a collection of OpenShift Container Platform features that are not enabled by default.

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

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



WARNING

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

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

- **AdditionalRoutingCapabilities**
- **AdminNetworkPolicy**
- **AlibabaPlatform**
- **AutomatedEtcdBackup**
- **AWSClusterHostedDNS**
- **AWSServiceLBNetworkSecurityGroup**
- **AzureMultiDisk**
- **AzureWorkloadIdentity**
- **BootcNodeManagement**
- **BuildCSIVolumes**
- **ClusterMonitoringConfig**
- **ClusterVersionOperatorConfiguration**
- **ConsolePluginContentSecurityPolicy**
- **CPMSMachineNamePrefix**
- **DNSNameResolver**
- **DynamicResourceAllocation**
- **DyanmicServiceEndpointIBMCloud**
- **EtcdBackendQuota**
- **Example**

- **ExternalOIDC**
- **ExternalOIDCWithUIDAndExtraClaimMappings**
- **GatewayAPI**
- **GatewayAPIController**
- **GCPClusterHostedDNSInstall**
- **GCPCustomAPIEndpoints**
- **HighlyAvailableArbiter**
- **ImageModeStatusReporting**
- **ImageStreamImportMode**
- **ImageVolume**
- **IngressControllerDynamicConfigurationManager**
- **IngressControllerLBSubnetsAWS**
- **InsightsConfig**
- **InsightsConfigAPI**
- **InsightsOnDemandDataGather**
- **IrreconcilableMachineConfig**
- **KMSEncryptionProvider**
- **KMSv1**
- **MachineAPIMigration**
- **MachineConfigNodes**
- **ManagedBootImages**
- **ManagedBootImagesAWS**
- **ManagedBootImagesAzure**
- **ManagedBootImagesvSphere**
- **MaxUnavailableStatefulSet**
- **MetricsCollectionProfiles**
- **MinimumKubeletVersion**
- **MixedCPUsAllocation**

- **MultiDiskSetup**
- **MutatingAdmissionPolicy**
- **NetworkDiagnosticsConfig**
- **NetworkLiveMigration**
- **NetworkSegmentation**
- **NewOLM**
- **NewOLMCatalogdAPIV1Metas**
- **NewOLMOwnSingleNamespace**
- **NewOLMPreflightPermissionChecks**
- **NewOLMWebhookProviderOpenshiftServiceCA**
- **NodeSwap**
- **NoRegistryClusterOperations**
- **NutanixMultiSubnets**
- **OpenShiftPodSecurityAdmission**
- **OVNObservability**
- **PinnedImages**
- **PreconfiguredUDNAddresses**
- **ProcMountType**
- **RouteAdvertisements**
- **RouteExternalCertificate**
- **SELinuxMount**
- **ServiceAccountTokenNodeBinding**
- **SetEIPForNLBIngressController**
- **SignatureStores**
- **SigstoreImageVerification**
- **SigstoreImageVerificationPKI**
- **TranslateStreamCloseWebsocketRequests**
- **UserNamespacesPodSecurityStandards**

- **UserNamespacesSupport**
- **VolumeAttributesClass**
- **VolumeGroupSnapshot**
- **VSphereConfigurableMaxAllowedBlockVolumesPerNode**
- **VSphereHostVMGroupZonal**
- **VSphereMixedNodeEnv**
- **VSphereMultiDisk**
- **VSphereMultiNetworks**
- **VSphereMultiVCenters**
- **TwoNodeOpenShiftClusterWithFencing**

See the *Additional resources* sections for information on some of these features.

4.11.2. Enabling feature sets using the web console

You can use the OpenShift Container Platform web console to enable feature sets for all of the nodes in a cluster by editing the **FeatureGate** custom resource (CR). Completing this task enables non-default features in your cluster.

Procedure

1. In the OpenShift Container Platform web console, switch to the **Administration → Custom Resource Definitions** page.
2. On the **Custom Resource Definitions** page, click **FeatureGate**.
3. On the **Custom Resource Definition Details** page, click the **Instances** tab.
4. Click the **cluster** feature gate, then click the **YAML** tab.
5. Edit the **cluster** instance to add specific feature sets:



WARNING

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

Sample Feature Gate custom resource

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

```
kind: FeatureGate
metadata:
  name: cluster 1
# ...
spec:
  featureSet: TechPreviewNoUpgrade 2
```

where:

metadata.name

Specifies the name of the **FeatureGate** CR. You must specify **cluster** for the name.

spec.featureSet

Specifies the feature set that you want to enable:

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

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

Verification

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

1. From the **Administrator** perspective in the web console, navigate to **Compute → Nodes**.
2. Select a node.
3. In the **Node details** page, click **Terminal**.
4. In the terminal window, change your root directory to **/host**:

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

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

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

Sample output

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

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



NOTE

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

4.11.3. Enabling feature sets using the CLI

You can use the OpenShift CLI (**oc**) to enable feature sets for all of the nodes in a cluster by editing the **FeatureGate** custom resource (CR). Completing this task enables non-default features in your cluster.

Prerequisites

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

Procedure

- Edit the **FeatureGate** CR named **cluster**:

```
$ oc edit featuregate cluster
```



WARNING

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

Sample FeatureGate custom resource

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

where:

metadata.name

Specifies the name of the **FeatureGate** CR. This must be **cluster**.

spec.featureSet

Specifies the feature set that you want to enable:

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

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

Verification

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

1. From the **Administrator** perspective in the web console, navigate to **Compute → Nodes**.
2. Select a node.
3. In the **Node details** page, click **Terminal**.
4. In the terminal window, change your root directory to **/host**:

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

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

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

Sample output

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

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



NOTE

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

4.12. ETCD TASKS

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



NOTE

If you deployed a bare-metal cluster, you can scale the cluster up to 5 nodes as part of your post-installation tasks. For more information, see [Node scaling for etcd](#).

4.12.1. 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. You must have these keys to restore from an etcd backup.



NOTE

Etcd encryption only encrypts values, not keys. Resource types, namespaces, and object names are unencrypted.

If etcd encryption is enabled during a backup, the **`static_kubernetes_<datetimestamp>.tar.gz`** file contains the encryption keys for the etcd snapshot. For security reasons, store this file separately from the etcd snapshot. However, this file is required to restore a previous state of etcd from the respective etcd snapshot.

4.12.2. Supported encryption types

The following encryption types are supported for encrypting etcd data in OpenShift Container Platform:

AES-CBC

Uses AES-CBC with PKCS#7 padding and a 32 byte key to perform the encryption. The encryption keys are rotated weekly.

AES-GCM

Uses AES-GCM with a random nonce and a 32 byte key to perform the encryption. The encryption keys are rotated weekly.

4.12.3. Enabling etcd encryption

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



WARNING

Do not back up etcd resources until the initial encryption process is completed. If the encryption process is not completed, the backup might be only partially encrypted.

After you enable etcd encryption, several changes can occur:

- The etcd encryption might affect the memory consumption of a few resources.
- You might notice a transient affect on backup performance because the leader must serve the backup.
- A disk I/O can affect the node that receives the backup state.

You can encrypt the etcd database in either AES-GCM or AES-CBC encryption.



NOTE

To migrate your etcd database from one encryption type to the other, you can modify the API server's **spec.encryption.type** field. Migration of the etcd data to the new encryption type occurs automatically.

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 **spec.encryption.type** field to **aesgcm** or **aescbc**:

```
spec:
  encryption:
    type: aesgcm 1
```

- 1 1 Set to **aesgcm** for AES-GCM encryption or **aescbc** for AES-CBC 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 the etcd database.

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.

4.12.4. 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.

4.12.5. Backing up etcd data

Follow these steps to back up etcd data by creating an etcd snapshot and backing up the resources for the static pods. This backup can be saved and used at a later time if you need to restore etcd.



IMPORTANT

Only save a backup from a single control plane host. Do not take a backup from each control plane host in the cluster.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have checked whether the cluster-wide proxy is enabled.

TIP

You can check whether the proxy is enabled by reviewing the output of **oc get proxy cluster -o yaml**. The proxy is enabled if the **httpProxy**, **httpsProxy**, and **noProxy** fields have values set.

Procedure

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

```
$ oc debug --as-root node/<node_name>
```

2. Change your root directory to **/host** in the debug shell:

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

3. If the cluster-wide proxy is enabled, export the **NO_PROXY**, **HTTP_PROXY**, and **HTTPS_PROXY** environment variables by running the following commands:

```
$ export HTTP_PROXY=http://<your_proxy.example.com>:8080
```

```
$ export HTTPS_PROXY=https://<your_proxy.example.com>:8080
```

```
$ export NO_PROXY=<example.com>
```

4. Run the **cluster-backup.sh** script in the debug shell and pass in the location to save the backup to.

TIP

The **cluster-backup.sh** script is maintained as a component of the etcd Cluster Operator and is a wrapper around the **etcdctl snapshot save** command.

```
sh-4.4# /usr/local/bin/cluster-backup.sh /home/core/assets/backup
```

Example script output

```
found latest kube-apiserver: /etc/kubernetes/static-pod-resources/kube-apiserver-pod-6
found latest kube-controller-manager: /etc/kubernetes/static-pod-resources/kube-controller-
manager-pod-7
found latest kube-scheduler: /etc/kubernetes/static-pod-resources/kube-scheduler-pod-6
found latest etcd: /etc/kubernetes/static-pod-resources/etcd-pod-3
ede95fe6b88b87ba86a03c15e669fb4aa5bf0991c180d3c6895ce72eaade54a1
etcdctl version: 3.4.14
API version: 3.4
{"level":"info","ts":1624647639.0188997,"caller":"snapshot/v3_snapshot.go:119","msg":"created
```

```
temporary db file","path":"/home/core/assets/backup/snapshot_2021-06-25_190035.db.part"}
{"level":"info","ts":"2021-06-25T19:00:39.030Z","caller":"clientv3/maintenance.go:200","msg":"opened snapshot stream; downloading"}
{"level":"info","ts":1624647639.0301006,"caller":"snapshot/v3_snapshot.go:127","msg":"fetching snapshot","endpoint":"https://10.0.0.5:2379"}
{"level":"info","ts":"2021-06-25T19:00:40.215Z","caller":"clientv3/maintenance.go:208","msg":"completed snapshot read; closing"}
{"level":"info","ts":1624647640.6032252,"caller":"snapshot/v3_snapshot.go:142","msg":"fetched snapshot","endpoint":"https://10.0.0.5:2379","size":"114 MB","took":1.584090459}
{"level":"info","ts":1624647640.6047094,"caller":"snapshot/v3_snapshot.go:152","msg":"saved","path":"/home/core/assets/backup/snapshot_2021-06-25_190035.db"}
Snapshot saved at /home/core/assets/backup/snapshot_2021-06-25_190035.db
{"hash":3866667823,"revision":31407,"totalKey":12828,"totalSize":114446336}
snapshot db and kube resources are successfully saved to /home/core/assets/backup
```

In this example, two files are created in the **/home/core/assets/backup/** directory on the control plane host:

- **snapshot_<datetimestamp>.db**: This file is the etcd snapshot. The **cluster-backup.sh** script confirms its validity.
- **static_kubernetes_<datetimestamp>.tar.gz**: This file contains the resources for the static pods. If etcd encryption is enabled, it also contains the encryption keys for the etcd snapshot.



NOTE

If etcd encryption is enabled, it is recommended to store this second file separately from the etcd snapshot for security reasons. However, this file is required to restore from the etcd snapshot.

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

4.12.6. Defragmenting etcd data

For large and dense clusters, etcd can suffer from poor performance if the keyspace grows too large and exceeds the space quota. Periodically maintain and defragment etcd to free up space in the data store. Monitor Prometheus for etcd metrics and defragment it when required; otherwise, etcd can raise a cluster-wide alarm that puts the cluster into a maintenance mode that accepts only key reads and deletes.

Monitor these key metrics:

- **etcd_server_quota_backend_bytes**, which is the current quota limit
- **etcd_mvcc_db_total_size_in_use_in_bytes**, which indicates the actual database usage after a history compaction
- **etcd_mvcc_db_total_size_in_bytes**, which shows the database size, including free space waiting for defragmentation

Defragment etcd data to reclaim disk space after events that cause disk fragmentation, such as etcd history compaction.

History compaction is performed automatically every five minutes and leaves gaps in the back-end database. This fragmented space is available for use by etcd, but is not available to the host file system. You must defragment etcd to make this space available to the host file system.

Defragmentation occurs automatically, but you can also trigger it manually.



NOTE

Automatic defragmentation is good for most cases, because the etcd operator uses cluster information to determine the most efficient operation for the user.

4.12.6.1. Automatic defragmentation

The etcd Operator automatically defragments disks. No manual intervention is needed.

Verify that the defragmentation process is successful by viewing one of these logs:

- etcd logs
- cluster-etcd-operator pod
- operator status error log



WARNING

Automatic defragmentation can cause leader election failure in various OpenShift core components, such as the Kubernetes controller manager, which triggers a restart of the failing component. The restart is harmless and either triggers failover to the next running instance or the component resumes work again after the restart.

Example log output for successful defragmentation

```
etcd member has been defragmented: <member_name>, memberID: <member_id>
```

Example log output for unsuccessful defragmentation

```
failed defrag on member: <member_name>, memberID: <member_id>: <error_message>
```

4.12.6.2. Manual defragmentation

A Prometheus alert indicates when you need to use manual defragmentation. The alert is displayed in two cases:

- When etcd uses more than 50% of its available space for more than 10 minutes

- When etcd is actively using less than 50% of its total database size for more than 10 minutes

You can also determine whether defragmentation is needed by checking the etcd database size in MB that will be freed by defragmentation with the PromQL expression:

`(etcd_mvcc_db_total_size_in_bytes - etcd_mvcc_db_total_size_in_use_in_bytes)/1024/1024`



WARNING

Defragmenting etcd is a blocking action. The etcd member will not respond until defragmentation is complete. For this reason, wait at least one minute between defragmentation actions on each of the pods to allow the cluster to recover.

Follow this procedure to defragment etcd data on each etcd member.

Prerequisites

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

Procedure

1. Determine which etcd member is the leader, because the leader should be defragmented last.

- a. Get the list of etcd pods:

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd -o wide
```

Example output

```
etcd-ip-10-0-159-225.example.redhat.com      3/3   Running   0      175m
10.0.159.225 ip-10-0-159-225.example.redhat.com <none>   <none>
etcd-ip-10-0-191-37.example.redhat.com      3/3   Running   0      173m
10.0.191.37 ip-10-0-191-37.example.redhat.com <none>   <none>
etcd-ip-10-0-199-170.example.redhat.com     3/3   Running   0      176m
10.0.199.170 ip-10-0-199-170.example.redhat.com <none>   <none>
```

- b. Choose a pod and run the following command to determine which etcd member is the leader:

```
$ oc rsh -n openshift-etcd etcd-ip-10-0-159-225.example.redhat.com etcdctl endpoint
status --cluster -w table
```

Example output

```
Defaulting container name to etcdctl.
Use 'oc describe pod/etcd-ip-10-0-159-225.example.redhat.com -n openshift-etcd' to see
all of the containers in this pod.
+-----+-----+-----+-----+-----+-----+
|      ENDPOINT      | ID   | VERSION | DB SIZE | IS LEADER | IS LEARNER |
+-----+-----+-----+-----+-----+-----+
| 10.0.159.225       | 1     | 3.3.10   | 100MB   | true      | false      |
| 10.0.191.37        | 2     | 3.3.10   | 100MB   | false     | false      |
| 10.0.199.170       | 3     | 3.3.10   | 100MB   | false     | false      |
```

```

RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+-----+-----+-----+-----+-----+
+-----+-----+-----+
| https://10.0.191.37:2379 | 251cd44483d811c3 | 3.5.9 | 104 MB | false | false |
7 | 91624 | 91624 | |
| https://10.0.159.225:2379 | 264c7c58ecbdabee | 3.5.9 | 104 MB | false | false |
7 | 91624 | 91624 | |
| https://10.0.199.170:2379 | 9ac311f93915cc79 | 3.5.9 | 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 | 3.5.9 | 104 MB | false | false |
7 | 91624 | 91624 | |
| https://10.0.159.225:2379 | 264c7c58ecbdabee | 3.5.9 | 41 MB | false | false |
7 | 91624 | 91624 | |

```

1


```
| https://10.0.199.170:2379 | 9ac311f93915cc79 | 3.5.9 | 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
```

4.12.7. Restoring to a previous cluster state for more than one node

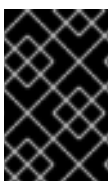
You can use a saved etcd backup to restore a previous cluster state or restore a cluster that has lost the majority of control plane hosts.

For high availability (HA) clusters, a three-node HA cluster requires you to shut down etcd on two hosts to avoid a cluster split. On four-node and five-node HA clusters, you must shut down three hosts. Quorum requires a simple majority of nodes. The minimum number of nodes required for quorum on a three-node HA cluster is two. On four-node and five-node HA clusters, the minimum number of nodes required for quorum is three. If you start a new cluster from backup on your recovery host, the other etcd members might still be able to form quorum and continue service.



NOTE

If your cluster uses a control plane machine set, see "Recovering a degraded etcd Operator" in "Troubleshooting the control plane machine set" for an etcd recovery procedure. For OpenShift Container Platform on a single node, see "Restoring to a previous cluster state for a single node".

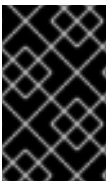


IMPORTANT

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 4.20.2 cluster must use an etcd backup that was taken from 4.20.2.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role through a certificate-based **kubeconfig** file, like the one that was used during installation.
- A healthy control plane host to use as the recovery host.
- You have 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_kubernetes_<timestamp>.tar.gz**.
- Nodes must be accessible or bootable.

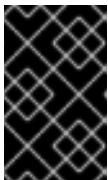


IMPORTANT

For non-recovery control plane nodes, it is not required to establish SSH connectivity or to stop the static pods. You can delete and re-create 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 run the restore operation on.
2. Establish SSH connectivity to each of the control plane nodes, including the recovery host. **kube-apiserver** 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. Using SSH, connect to each control plane node and run the following command to disable etcd:

```
$ sudo -E /usr/local/bin/disable-etcd.sh
```

4. 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.
5. Use SSH to connect to the recovery host and restore the cluster from a previous backup by running the following command:

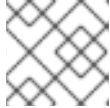
```
$ sudo -E /usr/local/bin/cluster-restore.sh /home/core/<etcd-backup-directory>
```

6. Exit the SSH session.
7. Once the API responds, turn off the etcd Operator quorum guard by running the following command:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides": {"useUnsupportedUnsafeNonHANonProductionUnstableEtcd": true}}}'
```

8. Monitor the recovery progress of the control plane by running the following command:

```
$ oc adm wait-for-stable-cluster
```



NOTE

It can take up to 15 minutes for the control plane to recover.

9. Once recovered, enable the quorum guard by running the following command:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides": null}}'
```

Troubleshooting

If you see no progress rolling out the etcd static pods, you can force redeployment from the **cluster-etcd-operator** by running the following command:

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

Additional resources

- [Recommended etcd practices](#)
- [Installing a user-provisioned cluster on bare metal](#)
- [Replacing a bare-metal control plane node](#)

4.12.8. 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. 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*).

4.13. POD DISRUPTION BUDGETS

Understand and configure pod disruption budgets.

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

A *pod disruption budget* allows the specification of safety constraints on pods during operations, such as draining a node for maintenance.

PodDisruptionBudget is an API object that specifies the minimum number or percentage of replicas that must be up at a time. Setting these in projects can be helpful during node maintenance (such as scaling a cluster down or a cluster upgrade) and is only honored on voluntary evictions (not on node failures).

A **PodDisruptionBudget** object's configuration consists of the following key parts:

- A label selector, which is a label query over a set of pods.
- An availability level, which specifies the minimum number of pods that must be available simultaneously, either:
 - **minAvailable** is the number of pods must always be available, even during a disruption.
 - **maxUnavailable** is the number of pods can be unavailable during a disruption.

**NOTE**

Available refers to the number of pods that has condition **Ready=True**. **Ready=True** refers to the pod that is able to serve requests and should be added to the load balancing pools of all matching services.

A **maxUnavailable** of **0%** or **0** or a **minAvailable** of **100%** or equal to the number of replicas is permitted but can block nodes from being drained.

**WARNING**

The default setting for **maxUnavailable** is **1** for all the machine config pools in OpenShift Container Platform. It is recommended to not change this value and update one control plane node at a time. Do not change this value to **3** for the control plane pool.

You can check for pod disruption budgets across all projects with the following:

```
$ oc get poddisruptionbudget --all-namespaces
```

**NOTE**

The following example contains some values that are specific to OpenShift Container Platform on AWS.

Example output

NAMESPACE	NAME	MIN AVAILABLE	MAX UNAVAILABLE
ALLOWED DISRUPTIONS	AGE		
openshift-apiserver	openshift-apiserver-pdb	N/A	1
121m			
openshift-cloud-controller-manager	aws-cloud-controller-manager	1	N/A
125m			1
openshift-cloud-credential-operator	pod-identity-webhook	1	N/A
117m			1
openshift-cluster-csi-drivers	aws-ebs-csi-driver-controller-pdb	N/A	1
121m			
openshift-cluster-storage-operator	csi-snapshot-controller-pdb	N/A	1
122m			
openshift-cluster-storage-operator	csi-snapshot-webhook-pdb	N/A	1
122m			
openshift-console	console	N/A	1
116m			
#...			

The **PodDisruptionBudget** is considered healthy when there are at least **minAvailable** pods running in the system. Every pod above that limit can be evicted.

**NOTE**

Depending on your pod priority and preemption settings, lower-priority pods might be removed despite their pod disruption budget requirements.

4.13.2. Specifying the number of pods that must be up with pod disruption budgets

You can use a **PodDisruptionBudget** object to specify the minimum number or percentage of replicas that must be up at a time.

Procedure

To configure a pod disruption budget:

1. Create a YAML file with the an object definition similar to the following:

```
apiVersion: policy/v1 1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  minAvailable: 2 2
  selector: 3
    matchLabels:
      name: my-pod
```

- 1** **PodDisruptionBudget** is part of the **policy/v1** API group.
- 2** The minimum number of pods that must be available simultaneously. This can be either an integer or a string specifying a percentage, for example, **20%**.
- 3** A label query over a set of resources. The result of **matchLabels** and **matchExpressions** are logically conjoined. Leave this parameter blank, for example **selector {}**, to select all pods in the project.

Or:

```
apiVersion: policy/v1 1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  maxUnavailable: 25% 2
  selector: 3
    matchLabels:
      name: my-pod
```

- 1** **PodDisruptionBudget** is part of the **policy/v1** API group.
- 2** The maximum number of pods that can be unavailable simultaneously. This can be either an integer or a string specifying a percentage, for example, **20%**.
- 3** A label query over a set of resources. The result of **matchLabels** and **matchExpressions**

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

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

4.13.3. Specifying the eviction policy for unhealthy pods

When you use pod disruption budgets (PDBs) to specify how many pods must be available simultaneously, you can also define the criteria for how unhealthy pods are considered for eviction.

You can choose one of the following policies:

IfHealthyBudget

Running pods that are not yet healthy can be evicted only if the guarded application is not disrupted.

AlwaysAllow

Running pods that are not yet healthy can be evicted regardless of whether the criteria in the pod disruption budget is met. This policy can help evict malfunctioning applications, such as ones with pods stuck in the **CrashLoopBackOff** state or failing to report the **Ready** status.



NOTE

It is recommended to set the **unhealthyPodEvictionPolicy** field to **AlwaysAllow** in the **PodDisruptionBudget** object to support the eviction of misbehaving applications during a node drain. The default behavior is to wait for the application pods to become healthy before the drain can proceed.

Procedure

1. Create a YAML file that defines a **PodDisruptionBudget** object and specify the unhealthy pod eviction policy:

Example pod-disruption-budget.yaml file

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      name: my-pod
  unhealthyPodEvictionPolicy: AlwaysAllow 1
```

- 1 Choose either **IfHealthyBudget** or **AlwaysAllow** as the unhealthy pod eviction policy. The default is **IfHealthyBudget** when the **unhealthyPodEvictionPolicy** field is empty.

2. Create the **PodDisruptionBudget** object by running the following command:

```
$ oc create -f pod-disruption-budget.yaml
```

With a PDB that has the **AlwaysAllow** unhealthy pod eviction policy set, you can now drain nodes and evict the pods for a malfunctioning application guarded by this PDB.

Additional resources

- [Enabling features using feature gates](#)
- [Unhealthy Pod Eviction Policy](#) in the Kubernetes documentation

CHAPTER 5. POSTINSTALLATION 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 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.

5.1.1. 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](#).

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

Procedure

1. Extract the Ignition config file from the cluster by running the following command:

```
$ oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --to=- > worker.ign
```

2. Upload the **worker.ign** Ignition config file you exported from your cluster to your HTTP server. Note the URLs of these files.
3. You can validate that the ignition files are available on the URLs. The following example gets the Ignition config files for the compute node:

```
$ curl -k http://<HTTP_server>/worker.ign
```

4. You can access the ISO image for booting your new machine by running to following command:

```
RHCOS_VHD_ORIGIN_URL=$(oc -n openshift-machine-config-operator get
configmap/coreos-bootimages -o jsonpath='{.data.stream}' | jq -r '.architectures.
<architecture>.artifacts.metal.formats.iso.disk.location')
```

5. 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.
6. Boot the RHCOS ISO image without specifying any options, or interrupting the live boot sequence. Wait for the installer to boot into a shell prompt in the RHCOS live environment.



NOTE

You can interrupt the RHCOS installation boot process to add kernel arguments. However, for this ISO procedure you must use the **coreos-installer** command as outlined in the following steps, instead of adding kernel arguments.

7. Run the **coreos-installer** command and specify the options that meet your installation requirements. At a minimum, you must specify the URL that points to the Ignition config file for the node type, and the device that you are installing to:

```
$ sudo coreos-installer install --ignition-url=http://<HTTP_server>/<node_type>.ign <device>
--ignition-hash=sha512-<digest> 1 2
```

- 1 You must run the **coreos-installer** command by using **sudo**, because the **core** user does not have the required root privileges to perform the installation.
- 2 The **--ignition-hash** option is required when the Ignition config file is obtained through an HTTP URL to validate the authenticity of the Ignition config file on the cluster node. **<digest>** is the Ignition config file SHA512 digest obtained in a preceding step.



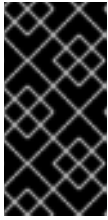
NOTE

If you want to provide your Ignition config files through an HTTPS server that uses TLS, you can add the internal certificate authority (CA) to the system trust store before running **coreos-installer**.

The following example initializes a compute node installation to the **/dev/sda** device. The Ignition config file for the compute node is obtained from an HTTP web server with the IP address 192.168.1.2:

```
$ sudo coreos-installer install --ignition-
url=http://192.168.1.2:80/installation_directory/worker.ign /dev/sda --ignition-hash=sha512-
a5a2d43879223273c9b60af66b44202a1d1248fc01cf156c46d4a79f552b6bad47bc8cc78ddf011
6e80c59d2ea9e32ba53bc807afbca581aa059311def2c3e3b
```

8. Monitor the progress of the RHCOS installation on the console of the machine.



IMPORTANT

Ensure that the installation is successful on each node before commencing with the OpenShift Container Platform installation. Observing the installation process can also help to determine the cause of RHCOS installation issues that might arise.

9. Continue to create more compute machines for your cluster.

5.1.3. Creating 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.

**NOTE**

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 (**x86_64** + **aarch64**):

```
kernel http://<HTTP_server>/rhcos-<version>-live-kernel-<architecture> initrd=main
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign 1 2
initrd --name main http://<HTTP_server>/rhcos-<version>-live-initramfs.
<architecture>.img 3
boot
```

- 1** Specify the 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.live.rootfs_url** parameter value is the location of the **rootfs** file, and the **coreos.inst.ignition_url** parameter value is the location of the worker Ignition config file.
- 2** If you use multiple NICs, specify a single interface in the **ip** option. For example, to use DHCP on a NIC that is named **eno1**, set **ip=eno1:dhcp**.
- 3** Specify the location of the **initramfs** file that you uploaded to your HTTP server.

**NOTE**

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?](#) and "Enabling the serial console for PXE and ISO installation" in the "Advanced RHCOS installation configuration" section.

**NOTE**

To network boot the CoreOS **kernel** on **aarch64** architecture, you need to use a version of iPXE build with the **IMAGE_GZIP** option enabled. See [IMAGE_GZIP option in iPXE](#).

- For PXE (with UEFI and GRUB as second stage) on **aarch64**:

```
menuentry 'Install CoreOS' {
    linux rhcos-<version>-live-kernel-<architecture>
```

```
coreos.live.rootfs_url=http://<HTTP_server>/rhcos-<version>-live-rootfs.
<architecture>.img coreos.inst.install_dev=/dev/sda
coreos.inst.ignition_url=http://<HTTP_server>/worker.ign
  initrd rhcos-<version>-live-initramfs.<architecture>.img
}
```

- 1 Specify the locations of the RHCOS files that you uploaded to your HTTP/TFTP server. The **kernel** parameter value is the location of the **kernel** file on your TFTP server. The **coreos.live.rootfs_url** parameter value is the location of the **rootfs** file, and the **coreos.inst.ignition_url** parameter value is the location of the worker Ignition config file on your HTTP Server.
- 2 If you use multiple NICs, specify a single interface in the **ip** option. For example, to use DHCP on a NIC that is named **eno1**, set **ip=eno1:dhcp**.
- 3 Specify the location of the **initramfs** file that you uploaded to your TFTP server.

2. Use the PXE or iPXE infrastructure to create the required compute machines for your cluster.

5.1.4. 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   v1.33.4
master-1  Ready    master   63m   v1.33.4
master-2  Ready    master   64m   v1.33.4
```

The output lists all of the machines that you created.



NOTE

The preceding output might not include the compute nodes, also known as worker nodes, until some CSRs are approved.

2. Review the pending CSRs and ensure that you see the client requests with the **Pending** or **Approved** status for each machine that you added to the cluster:

```
$ oc get csr
```

Example output

```
NAME      AGE  REQUESTOR                                     CONDITION
csr-8b2br  15m  system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper  Pending
csr-8vnps  15m  system:serviceaccount:openshift-machine-config-operator:node-
bootstrapper  Pending
...
```

In this example, two machines are joining the cluster. You might see more approved CSRs in the list.

- 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.

4. Now that your client requests are approved, you must review the server requests for each machine that you added to the cluster:

```
$ oc get csr
```

Example output

```
NAME      AGE   REQUESTOR                                CONDITION
csr-bfd72  5m26s system:node:ip-10-0-50-126.us-east-2.compute.internal
Pending
csr-c57lv  5m26s system:node:ip-10-0-95-157.us-east-2.compute.internal
Pending
...
```

5. If the remaining CSRs are not approved, and are in the **Pending** status, approve the CSRs for your cluster machines:

- To approve them individually, run the following command for each valid CSR:

```
$ oc adm certificate approve <csr_name> 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  v1.33.4
master-1  Ready   master  73m  v1.33.4
master-2  Ready   master  74m  v1.33.4
worker-0  Ready   worker  11m  v1.33.4
worker-1  Ready   worker  11m  v1.33.4
```

**NOTE**

It can take a few minutes after approval of the server CSRs for the machines to transition to the **Ready** status.

Additional information

- [Certificate Signing Requests](#)

5.1.5. Adding a new RHCOS worker node with a custom `/var` partition in AWS

OpenShift Container Platform supports partitioning devices during installation by using machine configs that are processed during the bootstrap. However, if you use `/var` partitioning, the device name must be determined at installation and cannot be changed. You cannot add different instance types as nodes if they have a different device naming schema. For example, if you configured the `/var` partition with the default AWS device name for **m4.large** instances, **dev/xvdb**, you cannot directly add an AWS **m5.large** instance, as **m5.large** instances use a **/dev/nvme1n1** device by default. The device might fail to partition due to the different naming schema.

The procedure in this section shows how to add a new Red Hat Enterprise Linux CoreOS (RHCOS) compute node with an instance that uses a different device name from what was configured at installation. You create a custom user data secret and configure a new compute machine set. These steps are specific to an AWS cluster. The principles apply to other cloud deployments also. However, the device naming schema is different for other deployments and should be determined on a per-case basis.

Procedure

1. On a command line, change to the **openshift-machine-api** namespace:

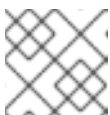
```
$ oc project openshift-machine-api
```

2. Create a new secret from the **worker-user-data** secret:

- a. Export the **userData** section of the secret to a text file:

```
$ oc get secret worker-user-data --template='{{index .data.userData | base64decode}}' |
jq > userData.txt
```

- b. Edit the text file to add the **storage**, **filesystems**, and **systemd** stanzas for the partitions you want to use for the new node. You can specify any [Ignition configuration parameters](#) as needed.



NOTE

Do not change the values in the **ignition** stanza.

```
{
  "ignition": {
    "config": {
      "merge": [
        {
          "source": "https:...."
        }
      ]
    },
    "security": {
      "tls": {
        "certificateAuthorities": [
```



```

    {
      "source": "data:text/plain;charset=utf-8;base64,.....=="
    }
  ]
},
"version": "3.2.0"
},
"storage": {
  "disks": [
    {
      "device": "/dev/nvme1n1", ❶
      "partitions": [
        {
          "label": "var",
          "sizeMiB": 50000, ❷
          "startMiB": 0 ❸
        }
      ]
    }
  ],
  "filesystems": [
    {
      "device": "/dev/disk/by-partlabel/var", ❹
      "format": "xfs", ❺
      "path": "/var" ❻
    }
  ]
},
"systemd": {
  "units": [ ❼
    {
      "contents": "[Unit]\nBefore=local-
fs.target\n[Mount]\nWhere=/var\nWhat=/dev/disk/by-
partlabel/var\nOptions=defaults,pquota\n[Install]\nWantedBy=local-fs.target\n",
      "enabled": true,
      "name": "var.mount"
    }
  ]
}
}

```

- ❶ Specifies an absolute path to the AWS block device.
- ❷ Specifies the size of the data partition in Mebibytes.
- ❸ Specifies the start of the partition in Mebibytes. When adding a data partition to the boot disk, a minimum value of 25000 MB (Mebibytes) is recommended. The root file system is automatically resized to fill all available space up to the specified offset. If no value is specified, or if the specified value is smaller than the recommended minimum, the resulting root file system will be too small, and future reinstalls of RHCOS might overwrite the beginning of the data partition.
- ❹ Specifies an absolute path to the **/var** partition.

- 5 Specifies the filesystem format.
- 6 Specifies the mount-point of the filesystem while Ignition is running relative to where the root filesystem will be mounted. This is not necessarily the same as where it should be mounted in the real root, but it is encouraged to make it the same.
- 7 Defines a systemd mount unit that mounts the **/dev/disk/by-partlabel/var** device to the **/var** partition.

- c. Extract the **disableTemplating** section from the **work-user-data** secret to a text file:

```
$ oc get secret worker-user-data --template='{{index .data.disableTemplating |
base64decode}}' | jq > disableTemplating.txt
```

- d. Create the new user data secret file from the two text files. This user data secret passes the additional node partition information in the **userData.txt** file to the newly created node.

```
$ oc create secret generic worker-user-data-x5 --from-file=userData=userData.txt --
from-file=disableTemplating=disableTemplating.txt
```

3. Create a new compute machine set for the new node:

- a. Create a new compute machine set YAML file, similar to the following, which is configured for AWS. Add the required partitions and the newly-created user data secret:

TIP

Use an existing compute machine set as a template and change the parameters as needed for the new node.

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: auto-52-92tf4
  name: worker-us-east-2-nvme1n1 1
  namespace: openshift-machine-api
spec:
  replicas: 1
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: auto-52-92tf4
      machine.openshift.io/cluster-api-machineset: auto-52-92tf4-worker-us-east-2b
  template:
    metadata:
      labels:
        machine.openshift.io/cluster-api-cluster: auto-52-92tf4
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: auto-52-92tf4-worker-us-east-2b
    spec:
      metadata: {}
      providerSpec:
```

```

value:
  ami:
    id: ami-0c2dbd95931a
  apiVersion: awsproviderconfig.openshift.io/v1beta1
  blockDevices:
    - DeviceName: /dev/nvme1n1 ❷
      ebs:
        encrypted: true
        iops: 0
        volumeSize: 120
        volumeType: gp2
    - DeviceName: /dev/nvme1n2 ❸
      ebs:
        encrypted: true
        iops: 0
        volumeSize: 50
        volumeType: gp2
  credentialsSecret:
    name: aws-cloud-credentials
  deviceIndex: 0
  iamInstanceProfile:
    id: auto-52-92tf4-worker-profile
  instanceType: m6i.large
  kind: AWSMachineProviderConfig
  metadata:
    creationTimestamp: null
  placement:
    availabilityZone: us-east-2b
    region: us-east-2
  securityGroups:
    - filters:
        - name: tag:Name
          values:
            - auto-52-92tf4-worker-sg
  subnet:
    id: subnet-07a90e5db1
  tags:
    - name: kubernetes.io/cluster/auto-52-92tf4
      value: owned
  userDataSecret:
    name: worker-user-data-x5 ❹

```

- ❶ Specifies a name for the new node.
- ❷ Specifies an absolute path to the AWS block device, here an encrypted EBS volume.
- ❸ Optional. Specifies an additional EBS volume.
- ❹ Specifies the user data secret file.

b. Create the compute machine set:

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

The machines might take a few moments to become available.

4. Verify that the new partition and nodes are created:

a. Verify that the compute machine set is created:

```
$ oc get machineset
```

Example output

NAME	DESIRED	CURRENT	READY	AVAILABLE	AGE
ci-ln-2675bt2-76ef8-bdgsc-worker-us-east-1a	1	1	1	1	124m
ci-ln-2675bt2-76ef8-bdgsc-worker-us-east-1b	2	2	2	2	124m
worker-us-east-2-nvme1n1	1	1	1	1	2m35s 1

1 This is the new compute machine set.

b. Verify that the new node is created:

```
$ oc get nodes
```

Example output

NAME	STATUS	ROLES	AGE	VERSION
ip-10-0-128-78.ec2.internal	Ready	worker	117m	v1.33.4
ip-10-0-146-113.ec2.internal	Ready	master	127m	v1.33.4
ip-10-0-153-35.ec2.internal	Ready	worker	118m	v1.33.4
ip-10-0-176-58.ec2.internal	Ready	master	126m	v1.33.4
ip-10-0-217-135.ec2.internal	Ready	worker	2m57s	v1.33.4 1
ip-10-0-225-248.ec2.internal	Ready	master	127m	v1.33.4
ip-10-0-245-59.ec2.internal	Ready	worker	116m	v1.33.4

1 This is new new node.

c. Verify that the custom **/var** partition is created on the new node:

```
$ oc debug node/<node-name> -- chroot /host lsblk
```

For example:

```
$ oc debug node/ip-10-0-217-135.ec2.internal -- chroot /host lsblk
```

Example output

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPOINT
nvme0n1	202:0	0	120G	0	disk	
nvme0n1p1	202:1	0	1M	0	part	
nvme0n1p2	202:2	0	127M	0	part	
nvme0n1p3	202:3	0	384M	0	part	/boot
`nvme0n1p4	202:4	0	119.5G	0	part	/sysroot
nvme1n1	202:16	0	50G	0	disk	
`nvme1n1p1	202:17	0	48.8G	0	part	/var 1

- 1 The **nvme1n1** device is mounted to the **/var** partition.

Additional resources

- For more information on how OpenShift Container Platform uses disk partitioning, see [Disk partitioning](#).

5.2. DEPLOYING MACHINE HEALTH CHECKS

Understand and deploy machine health checks.



IMPORTANT

You can use the advanced machine management and scaling capabilities only in clusters where the Machine API is operational. Clusters with user-provisioned infrastructure require additional validation and configuration to use the Machine API.

Clusters with the infrastructure platform type **none** cannot use the Machine API. This limitation applies even if the compute machines that are attached to the cluster are installed on a platform that supports the feature. This parameter cannot be changed after installation.

To view the platform type for your cluster, run the following command:

```
$ oc get infrastructure cluster -o jsonpath='{.status.platform}'
```

5.2.1. About machine health checks



NOTE

You can only apply a machine health check to machines that are managed by compute machine sets or control plane machine sets.

To monitor machine health, create a resource to define the configuration for a controller. Set a condition to check, such as staying in the **NotReady** status for five minutes or displaying a permanent condition in the node-problem-detector, and a label for the set of machines to monitor.

The controller that observes a **MachineHealthCheck** resource checks for the defined condition. If a machine fails the health check, the machine is automatically deleted and one is created to take its place. When a machine is deleted, you see a **machine deleted** event.

To limit disruptive impact of the machine deletion, the controller drains and deletes only one node at a time. If there are more unhealthy machines than the **maxUnhealthy** threshold allows for in the targeted pool of machines, remediation stops and therefore enables manual intervention.



NOTE

Consider the timeouts carefully, accounting for workloads and requirements.

- Long timeouts can result in long periods of downtime for the workload on the unhealthy machine.
- Too short timeouts can result in a remediation loop. For example, the timeout for checking the **NotReady** status must be long enough to allow the machine to complete the startup process.

To stop the check, remove the resource.

5.2.1.1. Limitations when deploying machine health checks

There are limitations to consider before deploying a machine health check:

- Only machines owned by a machine set are remediated by a machine health check.
- If the node for a machine is removed from the cluster, a machine health check considers the machine to be unhealthy and remediates it immediately.
- If the corresponding node for a machine does not join the cluster after the **nodeStartupTimeout**, the machine is remediated.
- A machine is remediated immediately if the **Machine** resource phase is **Failed**.

Additional resources

- [About control plane machine sets](#)

5.2.2. 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.

5.2.2.1. Short-circuiting machine health check remediation

Short-circuiting ensures that machine health checks remediate machines only when the cluster is healthy. Short-circuiting is configured through the **maxUnhealthy** field in the **MachineHealthCheck** resource.

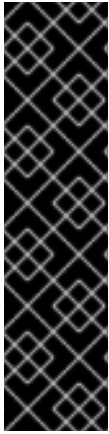
If the user defines a value for the **maxUnhealthy** field, before remediating any machines, the **MachineHealthCheck** compares the value of **maxUnhealthy** with the number of machines within its target pool that it has determined to be unhealthy. Remediation is not performed if the number of unhealthy machines exceeds the **maxUnhealthy** limit.



IMPORTANT

If **maxUnhealthy** is not set, the value defaults to **100%** and the machines are remediated regardless of the state of the cluster.

The appropriate **maxUnhealthy** value depends on the scale of the cluster you deploy and how many machines the **MachineHealthCheck** covers. For example, you can use the **maxUnhealthy** value to cover multiple compute machine sets across multiple availability zones so that if you lose an entire zone, your **maxUnhealthy** setting prevents further remediation within the cluster. In global Azure regions that do not have multiple availability zones, you can use availability sets to ensure high availability.



IMPORTANT

If you configure a **MachineHealthCheck** resource for the control plane, set the value of **maxUnhealthy** to **1**.

This configuration ensures that the machine health check takes no action when multiple control plane machines appear to be unhealthy. Multiple unhealthy control plane machines can indicate that the etcd cluster is degraded or that a scaling operation to replace a failed machine is in progress.

If the etcd cluster is degraded, manual intervention might be required. If a scaling operation is in progress, the machine health check should allow it to finish.

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

5.2.2.1.1. 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.

5.2.2.1.2. 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.

5.2.3. Creating a machine health check resource

You can create a **MachineHealthCheck** resource for machine sets in your cluster.



NOTE

You can only apply a machine health check to machines that are managed by compute machine sets or control plane machine sets.

Prerequisites

- Install the **oc** command-line interface.

Procedure

1. Create a **healthcheck.yml** file that contains the definition of your machine health check.
2. Apply the **healthcheck.yml** file to your cluster:

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

5.2.4. Scaling a compute machine set manually

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

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

Prerequisites

- Install an OpenShift Container Platform cluster and the **oc** command line.
- Log in to **oc** as a user with **cluster-admin** permission.

Procedure

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

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

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

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

```
$ oc get machines.machine.openshift.io -n openshift-machine-api
```

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

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

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

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

Or:

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

TIP

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

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

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

**IMPORTANT**

By default, the machine controller tries to drain the node that is backed by the machine until it succeeds. In some situations, such as with a misconfigured pod disruption budget, the drain operation might not be able to succeed. If the drain operation fails, the machine controller cannot proceed removing the machine.

You can skip draining the node by annotating **machine.openshift.io/exclude-node-draining** in a specific machine.

Verification

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

```
$ oc get machines.machine.openshift.io
```

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

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

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

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

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

5.3. RECOMMENDED NODE HOST PRACTICES

The OpenShift Container Platform node configuration file contains important options. For example, two parameters control the maximum number of pods that can be scheduled to a node: **podsPerCore** and **maxPods**.

When both options are in use, the lower of the two values limits the number of pods on a node. Exceeding these values can result in:

- Increased CPU utilization.
- Slow pod scheduling.
- Potential out-of-memory scenarios, depending on the amount of memory in the node.
- Exhausting the pool of IP addresses.
- Resource overcommitting, leading to poor user application performance.



IMPORTANT

In Kubernetes, a pod that is holding a single container actually uses two containers. The second container is used to set up networking prior to the actual container starting. Therefore, a system running 10 pods will actually have 20 containers running.



NOTE

Disk IOPS throttling from the cloud provider might have an impact on CRI-O and kubelet. They might get overloaded when there are large number of I/O intensive pods running on the nodes. It is recommended that you monitor the disk I/O on the nodes and use volumes with sufficient throughput for the workload.

The **podsPerCore** parameter sets the number of pods the node can run based on the number of processor cores on the node. For example, if **podsPerCore** is set to **10** on a node with 4 processor cores, the maximum number of pods allowed on the node will be **40**.

```
kubeletConfig:
  podsPerCore: 10
```

Setting **podsPerCore** to **0** disables this limit. The default is **0**. The value of the **podsPerCore** parameter cannot exceed the value of the **maxPods** parameter.

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

```
kubeletConfig:
  maxPods: 250
```

5.3.1. Creating a KubeletConfig CR to edit kubelet parameters

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



NOTE

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

Consider the following guidance:

- Edit an existing **KubeletConfig** CR to modify existing settings or add new settings, instead of creating a CR for each change. It is recommended that you create a CR only to modify a different machine config pool, or for changes that are intended to be temporary, so that you can revert the changes.
- Create one **KubeletConfig** CR for each machine config pool with all the config changes you want for that pool.
- As needed, create multiple **KubeletConfig** CRs with a limit of 10 per cluster. For the first **KubeletConfig** CR, the Machine Config Operator (MCO) creates a machine config appended with **kubelet**. With each subsequent CR, the controller creates another **kubelet** machine config with a numeric suffix. For example, if you have a **kubelet** machine config with a **-2** suffix, the next **kubelet** machine config is appended with **-3**.

NOTE

If you are applying a kubelet or container runtime config to a custom machine config pool, the custom role in the **machineConfigSelector** must match the name of the custom machine config pool.

For example, because the following custom machine config pool is named **infra**, the custom role must also be **infra**:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: infra
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,infra]}
# ...
```

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

NOTE

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

Example KubeletConfig CR

```
$ oc get kubeletconfig
```

NAME	AGE
set-kubelet-config	15m

Example showing a KubeletConfig machine config

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

```
...
99-worker-generated-kubelet-1          b5c5119de007945b6fe6fb215db3b8e2ceb12511  3.5.0
26m
...
```

The following procedure is an example to show how to configure the maximum number of pods per node, the maximum PIDs per node, and the maximum container log size 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-kubelet-config 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-kubelet-config
```

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. Configure the worker nodes as needed:

- a. Create a YAML file similar to the following that contains the kubelet configuration:



IMPORTANT

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

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-kubelet-config
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-kubelet-config 1
  kubeletConfig: 2
    podPidsLimit: 8192
    containerLogMaxSize: 50Mi
    maxPods: 500
```

1 Enter the label from the machine config pool.

2 Add the kubelet configuration. For example:

- Use **podPidsLimit** to set the maximum number of PIDs in any pod.
- Use **containerLogMaxSize** to set the maximum size of the container log file before it is rotated.
- 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-kubelet-config
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-kubelet-config
  kubeletConfig:
    maxPods: <pod_count>
    kubeAPIBurst: <burst_rate>
    kubeAPIQPS: <QPS>
```

- b. Update the machine config pool for workers with the label:

```
$ oc label machineconfigpool worker custom-kubelet=set-kubelet-config
```

- c. Create the **KubeletConfig** object:

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

Verification

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

```
$ oc get kubeletconfig
```

Example output

NAME	AGE
set-kubelet-config	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.

2. 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.

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

```
$ oc get kubeletconfigs set-kubelet-config -o yaml
```

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

```
spec:
  kubeletConfig:
    containerLogMaxSize: 50Mi
    maxPods: 500
    podPidsLimit: 8192
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-kubelet-config
status:
  conditions:
  - lastTransitionTime: "2021-06-30T17:04:07Z"
    message: Success
    status: "True"
    type: Success
```

5.3.2. Modifying the number of unavailable worker nodes

By default, only one machine is allowed to be unavailable when applying the kubelet-related configuration to the available worker nodes. For a large cluster, it can take a long time for the configuration change to be reflected. At any time, you can adjust the number of machines that are updating to speed up the process.

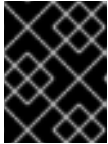
Procedure

1. Edit the **worker** machine config pool:

```
$ oc edit machineconfigpool worker
```

2. Add the **maxUnavailable** field and set the value:

```
spec:
  maxUnavailable: <node_count>
```

IMPORTANT

When setting the value, consider the number of worker nodes that can be unavailable without affecting the applications running on the cluster.

5.3.3. Control plane node sizing

The control plane node resource requirements depend on the number and type of nodes and objects in the cluster. The following control plane node size recommendations are based on the results of a control plane density focused testing, or *Cluster-density*. This test creates the following objects across a given number of namespaces:

- 1 image stream
- 1 build
- 5 deployments, with 2 pod replicas in a **sleep** state, mounting 4 secrets, 4 config maps, and 1 downward API volume each
- 5 services, each one pointing to the TCP/8080 and TCP/8443 ports of one of the previous deployments
- 1 route pointing to the first of the previous services
- 10 secrets containing 2048 random string characters
- 10 config maps containing 2048 random string characters

Number of worker nodes	Cluster-density (namespaces)	CPU cores	Memory (GB)
24	500	4	16
120	1000	8	32
252	4000	16, but 24 if using the OVN-Kubernetes network plug-in	64, but 128 if using the OVN-Kubernetes network plug-in
501, but untested with the OVN-Kubernetes network plug-in	4000	16	96

The data from the table above is based on an OpenShift Container Platform running on top of AWS, using r5.4xlarge instances as control-plane nodes and m5.2xlarge instances as worker nodes.

On a large and dense cluster with three control plane nodes, the CPU and memory usage will spike up when one of the nodes is stopped, rebooted, or fails. The failures can be due to unexpected issues with power, network, underlying infrastructure, or intentional cases where the cluster is restarted after shutting it down to save costs. The remaining two control plane nodes must handle the load in order to be highly available, which leads to increase in the resource usage. This is also expected during upgrades because the control plane nodes are cordoned, drained, and rebooted serially to apply the operating system updates, as well as the control plane Operators update. To avoid cascading failures, keep the

overall CPU and memory resource usage on the control plane nodes to at most 60% of all available capacity to handle the resource usage spikes. Increase the CPU and memory on the control plane nodes accordingly to avoid potential downtime due to lack of resources.

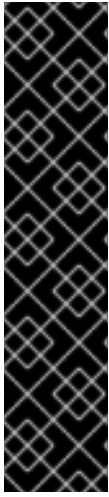


IMPORTANT

The node sizing varies depending on the number of nodes and object counts in the cluster. It also depends on whether the objects are actively being created on the cluster. During object creation, the control plane is more active in terms of resource usage compared to when the objects are in the **Running** phase.

Operator Lifecycle Manager (OLM) runs on the control plane nodes and its memory footprint depends on the number of namespaces and user installed operators that OLM needs to manage on the cluster. Control plane nodes need to be sized accordingly to avoid OOM kills. Following data points are based on the results from cluster maximums testing.

Number of namespaces	OLM memory at idle state (GB)	OLM memory with 5 user operators installed (GB)
500	0.823	1.7
1000	1.2	2.5
1500	1.7	3.2
2000	2	4.4
3000	2.7	5.6
4000	3.8	7.6
5000	4.2	9.02
6000	5.8	11.3
7000	6.6	12.9
8000	6.9	14.8
9000	8	17.7
10,000	9.9	21.6



IMPORTANT

You can modify the control plane node size in a running OpenShift Container Platform 4.20 cluster for the following configurations only:

- Clusters installed with a user-provisioned installation method.
- AWS clusters installed with an installer-provisioned infrastructure installation method.
- Clusters that use a control plane machine set to manage control plane machines.

For all other configurations, you must estimate your total node count and use the suggested control plane node size during installation.



NOTE

In OpenShift Container Platform 4.20, 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.

5.3.4. Setting up CPU Manager

To configure CPU manager, create a KubeletConfig custom resource (CR) and apply it to the desired set of nodes.

Procedure

1. Label a node by running the following command:

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

2. To enable CPU Manager for all compute nodes, edit the CR by running the following command:

```
# oc edit machineconfigpool worker
```

3. Add the **custom-kubelet: cpumanager-enabled** label to **metadata.labels** section.

```
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. This is the default policy.
- **static**. This policy allows containers in guaranteed pods with integer CPU requests. It also limits access to exclusive CPUs on the node. If **static**, you must use a lowercase **s**.

2Optional. Specify the CPU Manager reconcile frequency. The default is **5s**.

5. Create the dynamic kubelet config by running the following command:

```
# 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 by running the following command:

```
# 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 compute node for the updated **kubelet.conf** file by running the following command:

```
# oc debug node/perf-node.example.com
sh-4.2# cat /host/etc/kubernetes/kubelet.conf | grep cpuManager
```

Example output

```

cpuManagerPolicy: static 1
cpuManagerReconcilePeriod: 5s 2

```

1**cpuManagerPolicy** is defined when you create the **KubeletConfig** CR.**2****cpuManagerReconcilePeriod** is defined when you create the **KubeletConfig** CR.

8. Create a project by running the following command:

```
$ oc new-project <project_name>
```

9. 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:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: cpumanager
    image: gcr.io/google_containers/pause:3.2
    resources:
      requests:
        cpu: 1
        memory: "1G"
      limits:
        cpu: 1
        memory: "1G"
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  nodeSelector:
    cpumanager: "true"
```

10. Create the pod:

```
# oc create -f cpumanager-pod.yaml
```

Verification

1. Verify that the pod is scheduled to the node that you labeled by running the following command:

```
# 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

```

2. Verify that a CPU has been exclusively assigned to the pod by running the following command:

```
# oc describe node --selector='cpumanager=true' | grep -i cpumanager- -B2
```

Example output

NAMESPACE	NAME	CPU Requests	CPU Limits	Memory Requests	Memory Limits	Age
cpuman	cpumanager-mlrrz	1 (28%)	1 (28%)	1G (13%)	1G (13%)	27m

3. Verify that the **cgroups** are set up correctly. Get the process ID (PID) of the **pause** process by running the following commands:

```
# oc debug node/perf-node.example.com
```

```
sh-4.2# systemctl status | grep -B5 pause
```



NOTE

If the output returns multiple pause process entries, you must identify the correct pause process.

Example output

```

# └─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

```

4. Verify that pods of quality of service (QoS) tier **Guaranteed** are placed within the **kubepods.slice** subdirectory by running the following commands:

```
# cd /sys/fs/cgroup/kubepods.slice/kubepods-
pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice/crio-
b5437308f1ad1a7db0574c542bdf08563b865c0345c8e9585f8c0b0a655612c.scope
```

```
# for i in `ls cpuset.cpus cgroup.procs` ; do echo -n "$i "; cat $i ; done
```

**NOTE**

Pods of other QoS tiers end up in child **cgroups** of the parent **kubepods**.

Example output

```
cpuset.cpus 1
tasks 32706
```

5. Check the allowed CPU list for the task by running the following command:

```
# grep ^Cpus_allowed_list /proc/32706/status
```

Example output

```
Cpus_allowed_list: 1
```

6. Verify that another pod on the system cannot run on the core allocated for the **Guaranteed** pod. For example, to verify the pod in the **besteffort** QoS tier, run the following commands:

```
# cat /sys/fs/cgroup/kubepods.slice/kubepods-besteffort.slice/kubepods-besteffort-
podc494a073_6b77_11e9_98c0_06bba5c387ea.slice/crio-
c56982f57b75a2420947f0afc6cafe7534c5734efc34157525fa9abbf99e3849.scope/cpuset.cpus
```

```
# 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

5.4. HUGE PAGES

Understand and configure huge pages.

5.4.1. 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.

5.4.2. How huge pages are consumed by apps

Nodes must pre-allocate huge pages in order for the node to report its huge page capacity. A node can only pre-allocate huge pages for a single size.

Huge pages can be consumed through container-level resource requirements using the resource name **hugepages-<size>**, where size is the most compact binary notation using integer values supported on a particular node. For example, if a node supports 2048KiB page sizes, it exposes a schedulable resource **hugepages-2Mi**. Unlike CPU or memory, huge pages do not support over-commitment.

```
apiVersion: v1
kind: Pod
metadata:
  generateName: hugepages-volume-
```



```
spec:
  containers:
  - securityContext:
      privileged: true
    image: rhel7:latest
    command:
    - sleep
    - inf
    name: example
    volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
    resources:
      limits:
        hugepages-2Mi: 100Mi 1
        memory: "1Gi"
        cpu: "1"
    volumes:
    - name: hugepage
      emptyDir:
        medium: HugePages
```

- 1** Specify the amount of memory for **hugepages** as the exact amount to be allocated. Do not specify this value as the amount of memory for **hugepages** multiplied by the size of the page. For example, given a huge page size of 2MB, if you want to use 100MB of huge-page-backed RAM for your application, then you would allocate 50 huge pages. OpenShift Container Platform handles the math for you. As in the above example, you can specify **100MB** directly.

Allocating huge pages of a specific size

Some platforms support multiple huge page sizes. To allocate huge pages of a specific size, precede the huge pages boot command parameters with a huge page size selection parameter **hugepagesz=<size>**. The **<size>** value must be specified in bytes with an optional scale suffix [**kKmMgG**]. The default huge page size can be defined with the **default_hugepagesz=<size>** boot parameter.

Huge page requirements

- Huge page requests must equal the limits. This is the default if limits are specified, but requests are not.
- Huge pages are isolated at a pod scope. Container isolation is planned in a future iteration.
- **EmptyDir** volumes backed by huge pages must not consume more huge page memory than the pod request.
- Applications that consume huge pages via **shmget()** with **SHM_HUGETLB** must run with a supplemental group that matches *proc/sys/vm/hugetlb_shm_group*.

5.4.3. Configuring huge pages at boot time

Nodes must pre-allocate huge pages used in an OpenShift Container Platform cluster. There are two ways of reserving huge pages: at boot time and at run time. Reserving at boot time increases the possibility of success because the memory has not yet been significantly fragmented. The Node Tuning Operator currently supports boot time allocation of huge pages on specific nodes.

Procedure

To minimize node reboots, the order of the steps below needs to be followed:

1. Label all nodes that need the same huge pages setting by a label.

```
$ oc label node <node_using_hugepages> node-role.kubernetes.io/worker-hp=
```

2. Create a file with the following content and name it **hugepages-tuned-boottime.yaml**:

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: hugepages 1
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile: 2
  - data: |
      [main]
      summary=Boot time configuration for hugepages
      include=openshift-node
      [bootloader]
      cmdline_openshift_node_hugepages=hugepagesz=2M hugepages=50 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
```

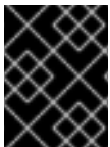


NOTE

The TuneD bootloader plugin only supports Red Hat Enterprise Linux CoreOS (RHCOS) worker nodes.

5.5. UNDERSTANDING DEVICE PLUGINS

The device plugin provides a consistent and portable solution to consume hardware devices across clusters. The device plugin provides support for these devices through an extension mechanism, which makes these devices available to Containers, provides health checks of these devices, and securely shares them.



IMPORTANT

OpenShift Container Platform supports the device plugin API, but the device plugin Containers are supported by individual vendors.

A device plugin is a gRPC service running on the nodes (external to the **kubelet**) that is responsible for managing specific hardware resources. Any device plugin must support following remote procedure calls (RPCs):

```

service DevicePlugin {
  // GetDevicePluginOptions returns options to be communicated with Device
  // Manager
  rpc GetDevicePluginOptions(Empty) returns (DevicePluginOptions) {}

  // ListAndWatch returns a stream of List of Devices
  // Whenever a Device state change or a Device disappears, ListAndWatch
  // returns the new list
  rpc ListAndWatch(Empty) returns (stream ListAndWatchResponse) {}

  // Allocate is called during container creation so that the Device
  // Plug-in can run device specific operations and instruct Kubelet
  // of the steps to make the Device available in the container
  rpc Allocate(AllocateRequest) returns (AllocateResponse) {}

  // PreStartcontainer is called, if indicated by Device Plug-in during

```

```

// registration phase, before each container start. Device plug-in
// can run device specific operations such as resetting the device
// before making devices available to the container
rpc PreStartcontainer(PreStartcontainerRequest) returns (PreStartcontainerResponse) {}
}

```

5.5.1. Example device plugins

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



NOTE

For easy device plugin reference implementation, there is a stub device plugin in the Device Manager code:

`vendor/k8s.io/kubernetes/pkg/kubelet/cm/deviceplugin/device_plugin_stub.go`.

5.5.2. Methods for deploying a device plugin

- Daemon sets are the recommended approach for device plugin deployments.
- Upon start, the device plugin will try to create a UNIX domain socket at `/var/lib/kubelet/device-plugin/` on the node to serve RPCs from Device Manager.
- Since device plugins must manage hardware resources, access to the host file system, as well as socket creation, they must be run in a privileged security context.
- More specific details regarding deployment steps can be found with each device plugin implementation.

5.5.3. Understanding the Device Manager

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

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



IMPORTANT

OpenShift Container Platform supports the device plugin API, but the device plugin Containers are supported by individual vendors.

Device Manager advertises devices as **Extended Resources**. User pods can consume devices, advertised by Device Manager, using the same **Limit/Request** mechanism, which is used for requesting any other **Extended Resource**.

Upon start, the device plugin registers itself with Device Manager invoking **Register** on the `/var/lib/kubelet/device-plugins/kubelet.sock` and starts a gRPC service at `/var/lib/kubelet/device-plugins/<plugin>.sock` for serving Device Manager requests.

Device Manager, while processing a new registration request, invokes **ListAndWatch** remote procedure call (RPC) at the device plugin service. In response, Device Manager gets a list of **Device** objects from the plugin over a gRPC stream. Device Manager will keep watching on the stream for new updates from the plugin. On the plugin side, the plugin will also keep the stream open and whenever there is a change in the state of any of the devices, a new device list is sent to the Device Manager over the same streaming connection.

While handling a new pod admission request, Kubelet passes requested **Extended Resources** to the Device Manager for device allocation. Device Manager checks in its database to verify if a corresponding plugin exists or not. If the plugin exists and there are free allocatable devices as well as per local cache, **Allocate** RPC is invoked at that particular device plugin.

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

5.5.4. Enabling Device Manager

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

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

1. Obtain the label associated with the static **MachineConfigPool** CRD for the type of node you want to configure by entering the following command. Perform one of the following steps:
 - a. View the machine config:

```
# oc describe machineconfig <name>
```

For example:

```
# oc describe machineconfig 00-worker
```

Example output

```
Name:      00-worker
Namespace:
Labels:    machineconfiguration.openshift.io/role=worker 1
```

1 Label required for the Device Manager.

Procedure

1. Create a custom resource (CR) for your configuration change.

Sample configuration for a Device Manager CR

```

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: devicemgr ❶
spec:
  machineConfigPoolSelector:
    matchLabels:
      machineconfiguration.openshift.io: devicemgr ❷
  kubeletConfig:
    feature-gates:
      - DevicePlugins=true ❸

```

- ❶ Assign a name to CR.
- ❷ Enter the label from the Machine Config Pool.
- ❸ Set **DevicePlugins** to 'true`.

2. Create the Device Manager:

```
$ oc create -f devicemgr.yaml
```

Example output

```
kubeletconfig.machineconfiguration.openshift.io/devicemgr created
```

3. Ensure that Device Manager was actually enabled by confirming that `/var/lib/kubelet/device-plugins/kubelet.sock` is created on the node. This is the UNIX domain socket on which the Device Manager gRPC server listens for new plugin registrations. This sock file is created when the Kubelet is started only if Device Manager is enabled.

5.6. TAINTS AND TOLERATIONS

Understand and work with taints and tolerations.

5.6.1. 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 to a node, the scheduler cannot place a pod on that node unless the pod can tolerate the taint.

Example taint in a node specification

```

apiVersion: v1
kind: Node
metadata:
  name: my-node
#...
spec:
  taints:

```

```
- effect: NoExecute
  key: key1
  value: value1
#...
```

Example toleration in a Pod spec

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
    tolerationSeconds: 3600
#...
```

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

Table 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.
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.

Parameter	Description						
effect	The effect is one of the following: <table> <tr> <td>NoSchedule ^[1]</td><td> - New pods that do not match the taint are not scheduled onto that node. - Existing pods on the node remain. </td></tr> <tr> <td>PreferNoSchedule</td><td> - New pods that do not match the taint might be scheduled onto that node, but the scheduler tries not to. - Existing pods on the node remain. </td></tr> <tr> <td>NoExecute</td><td> - New pods that do not match the taint cannot be scheduled onto that node. - Existing pods on the node that do not have a matching toleration are removed. </td></tr> </table>	NoSchedule ^[1]	- New pods that do not match the taint are not scheduled onto that node. - Existing pods on the node remain.	PreferNoSchedule	- 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	- 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]	- New pods that do not match the taint are not scheduled onto that node. - Existing pods on the node remain.						
PreferNoSchedule	- 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	- 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:

```

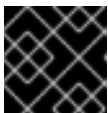
apiVersion: v1
kind: Node
metadata:
  annotations:
    machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-f76d1-v8jxv-master-0
    machineconfiguration.openshift.io/currentConfig: rendered-master-
cdc1ab7da414629332cc4c3926e6e59c
    name: my-node
  #...
spec:
  taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
  #...
```


A toleration matches a taint:

- If the **operator** parameter is set to **Equal**:
 - the **key** parameters are the same;
 - the **value** parameters are the same;
 - the **effect** parameters are the same.
- If the **operator** parameter is set to **Exists**:
 - the **key** parameters are the same;
 - the **effect** parameters are the same.

The following taints are built into OpenShift Container Platform:

- **node.kubernetes.io/not-ready**: The node is not ready. This corresponds to the node condition **Ready=False**.
- **node.kubernetes.io/unreachable**: The node is unreachable from the node controller. This corresponds to the node condition **Ready=Unknown**.
- **node.kubernetes.io/memory-pressure**: The node has memory pressure issues. This corresponds to the node condition **MemoryPressure=True**.
- **node.kubernetes.io/disk-pressure**: The node has disk pressure issues. This corresponds to the node condition **DiskPressure=True**.
- **node.kubernetes.io/network-unavailable**: The node network is unavailable.
- **node.kubernetes.io/unschedulable**: The node is unschedulable.
- **node.cloudprovider.kubernetes.io/uninitialized**: When the node controller is started with an external cloud provider, this taint is set on a node to mark it as unusable. After a controller from the cloud-controller-manager initializes this node, the kubelet removes this taint.
- **node.kubernetes.io/pid-pressure**: The node has pid pressure. This corresponds to the node condition **PIDPressure=True**.



IMPORTANT

OpenShift Container Platform does not set a default pid.available **evictionHard**.

5.6.2. Adding taints and tolerations

You add tolerations to pods and taints to nodes to allow the node to control which pods should or should not be scheduled on them. For existing pods and nodes, you should add the toleration to the pod first, then add the taint to the node to avoid pods being removed from the node before you can add the toleration.

Procedure

1. Add a toleration to a pod by editing the **Pod** spec to include a **tolerations** stanza:

Sample pod configuration file with an Equal operator

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
    - key: "key1" ❶
      value: "value1"
      operator: "Equal"
      effect: "NoExecute"
      tolerationSeconds: 3600 ❷
#...
```

❶ The toleration parameters, as described in the **Taint and toleration components** table.

❷ The **tolerationSeconds** parameter specifies how long a pod can remain bound to a node before being evicted.

For example:

Sample pod configuration file with an Exists operator

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
    - key: "key1"
      operator: "Exists" 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
  name: my-node
#...
spec:
  taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
#...
```

The tolerations on the pod match the taint on the node. A pod with either toleration can be scheduled onto **node1**.

5.6.3. Adding taints and tolerations using a compute machine set

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

Procedure

1. Add a toleration to a pod by editing the **Pod** spec to include a **tolerations** stanza:

Sample pod configuration file with **Equal** operator

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
    - key: "key1" ❶
      value: "value1"
      operator: "Equal"
      effect: "NoExecute"
      tolerationSeconds: 3600 ❷
#...
```

- 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

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
    - key: "key1"
      operator: "Exists"
      effect: "NoExecute"
      tolerationSeconds: 3600
#...
```

2. Add the taint to the **MachineSet** object:
 - a. Edit the **MachineSet** YAML for the nodes you want to taint or you can create a new **MachineSet** object:

```
$ oc edit machineset <machineset>
```

- b. Add the taint to the **spec.template.spec** section:

Example taint in a compute machine set specification

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: my-machineset
#...
spec:
  #...
  template:
    #...
    spec:
      taints:
        - effect: NoExecute
          key: key1
          value: value1
      #...
```

This example places a taint that has the key **key1**, value **value1**, and taint effect **NoExecute** on the nodes.

- c. Scale down the compute machine set to 0:

■

```
$ oc scale --replicas=0 machineset <machineset> -n openshift-machine-api
```

TIP

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

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 0
```

Wait for the machines to be removed.

- d. Scale up the compute machine set as needed:

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

Or:

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

Wait for the machines to start. The taint is added to the nodes associated with the **MachineSet** object.

5.6.4. Binding a user to a node using taints and tolerations

If you want to dedicate a set of nodes for exclusive use by a particular set of users, add a toleration to their pods. Then, add a corresponding taint to those nodes. The pods with the tolerations are allowed to use the tainted nodes or any other nodes in the cluster.

If you want ensure the pods are scheduled to only those tainted nodes, also add a label to the same set of nodes and add a node affinity to the pods so that the pods can only be scheduled onto nodes with that label.

Procedure

To configure a node so that users can use only that node:

1. Add a corresponding taint to those nodes:

For example:

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

TIP

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

```
kind: Node
apiVersion: v1
metadata:
  name: my-node
#...
spec:
  taints:
    - key: dedicated
      value: groupName
      effect: NoSchedule
#...
```

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

5.6.5. Controlling nodes with special hardware using taints and tolerations

In a cluster where a small subset of nodes have specialized hardware, you can use taints and tolerations to keep pods that do not need the specialized hardware off of those nodes, leaving the nodes for pods that do need the specialized hardware. You can also require pods that need specialized hardware to use specific nodes.

You can achieve this by adding a toleration to pods that need the special hardware and tainting the nodes that have the specialized hardware.

Procedure

To ensure nodes with specialized hardware are reserved for specific pods:

1. Add a toleration to pods that need the special hardware.

For example:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
    - key: "disktype"
      value: "ssd"
      operator: "Equal"
      effect: "NoSchedule"
      tolerationSeconds: 3600
#...
```

2. Taint the nodes that have the specialized hardware using one of the following commands:

```
$ oc adm taint nodes <node-name> disktype=ssd:NoSchedule
```

Or:

-

```
$ oc adm taint nodes <node-name> disktype=ssd:PreferNoSchedule
```

TIP

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

```
kind: Node
apiVersion: v1
metadata:
  name: my_node
#...
spec:
  taints:
    - key: disktype
      value: ssd
      effect: PreferNoSchedule
#...
```

5.6.6. Removing taints and tolerations

You can remove taints from nodes and tolerations from pods as needed. You should add the toleration to the pod first, then add the taint to the node to avoid pods being removed from the node before you can add the toleration.

Procedure

To remove taints and tolerations:

1. To remove a taint from a node:

```
$ oc adm taint nodes <node-name> <key>-
```

For example:

```
$ oc adm taint nodes ip-10-0-132-248.ec2.internal key1-
```

Example output

```
node/ip-10-0-132-248.ec2.internal untainted
```

2. To remove a toleration from a pod, edit the **Pod** spec to remove the toleration:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
    - key: "key2"
      operator: "Exists"
```

```
effect: "NoExecute"  
tolerationSeconds: 3600  
#...
```

5.7. TOPOLOGY MANAGER

Understand and work with Topology Manager.

5.7.1. Topology Manager policies

Topology Manager aligns **Pod** resources of all Quality of Service (QoS) classes by collecting topology hints from Hint Providers, such as CPU Manager and Device Manager, and using the collected hints to align the **Pod** resources.

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

none policy

This is the default policy and does not perform any topology alignment.

best-effort policy

For each container in a pod with the **best-effort** topology management policy, kubelet tries to align all the required resources on a NUMA node according to the preferred NUMA node affinity for that container. Even if the allocation is not possible due to insufficient resources, the Topology Manager still admits the pod but the allocation is shared with other NUMA nodes.

restricted policy

For each container in a pod with the **restricted** topology management policy, kubelet determines the theoretical minimum number of NUMA nodes that can fulfill the request. If the actual allocation requires more than that number of NUMA nodes, the Topology Manager rejects the admission, placing the pod in a **Terminated** state. If the number of NUMA nodes can fulfill the request, the Topology Manager admits the pod and the pod starts running.

single-numa-node policy

For each container in a pod with the **single-numa-node** topology management policy, kubelet admits the pod if all the resources required by the pod can be allocated on the same NUMA 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.

5.7.2. Setting up Topology Manager

To use Topology Manager, you must configure an allocation policy in the **KubeletConfig** custom resource (CR) named **cpumanager-enabled**. This file might exist if you have set up CPU Manager. If the file does not exist, you can create the file.

Prerequisites

- Configure the CPU Manager policy to be **static**.

Procedure

To activate Topology Manager:

1. Configure the Topology Manager allocation policy in the custom resource.

—


```
$ oc edit KubeletConfig cpumanager-enabled
```

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cpumanager-enabled
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: cpumanager-enabled
  kubeletConfig:
    cpuManagerPolicy: static 1
    cpuManagerReconcilePeriod: 5s
    topologyManagerPolicy: single-numa-node 2
```

- 1** This parameter must be **static** with a lowercase **s**.
- 2** Specify your selected Topology Manager allocation policy. Here, the policy is **single-numa-node**. Acceptable values are: **default**, **best-effort**, **restricted**, **single-numa-node**.

5.7.3. Pod interactions with Topology Manager policies

The example **Pod** specs 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 process all the pods and it enforces resource alignment only for the **Guaranteed** QoS **Pod** specification. When the Topology Manager policy is set to **none**, the relevant containers are pinned to any available CPU without considering NUMA affinity. This is the default behavior and it does not optimize for performance-sensitive workloads. Other values enable the use of topology awareness information from device plugins core resources, such as CPU and memory. The Topology Manager attempts to align the CPU, memory, and device allocations according to the topology of the node when the policy is set to other values than **none**. For more information about the available values, see *Topology Manager policies*.

The following example pod runs in the **Guaranteed** QoS class because requests are equal to limits.

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
      requests:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
```

Topology Manager would consider this pod. The Topology Manager would consult the Hint Providers, which are the CPU Manager, the Device Manager, and the Memory Manager, to get topology hints for the pod.

Topology Manager will use this information to store the best topology for this container. In the case of this pod, CPU Manager and Device Manager will use this stored information at the resource allocation stage.

5.8. RESOURCE REQUESTS AND OVERCOMMITMENT

You can use resource requests in an overcommitted environment help you ensure that your cluster is properly configured.

For each compute resource, a container can 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, where quota and hard limits refer to resource limits, which can be set higher than requested resources. The difference between the request and the limit determines the level of overcommit. For example, if a container is given a memory request of 1Gi and a memory limit of 2Gi, the container is scheduled based on the 1Gi request being available on the node, but could use up to 2Gi; so it is 100% overcommitted.

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

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

The Operator modifies the ratio between the requests and limits that are set on developer containers. In conjunction with a per-project limit range that specifies limits and defaults, you can achieve the desired level of overcommit.

You must install the Cluster Resource Override Operator by using the OpenShift Container Platform console or CLI as shown in the following sections. After you deploy the Cluster Resource Override Operator, the Operator modifies all new pods in specific namespaces. The Operator does not edit pods that existed before you deployed the Operator.

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
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50
      cpuRequestToLimitPercent: 25
      limitCPUMemoryPercent: 200
# ...
```

where:

metadata.name

Specifies a name for the object. The name must be **cluster**.

spec.podResourceOverride.spec.memoryRequestToLimitPercent

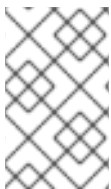
If a container memory limit has been specified or defaulted, the memory request is overridden to this percentage of the limit, between 1-100. The default is 50.

spec.podResourceOverride.spec.cpuRequestToLimitPercent

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.

spec.podResourceOverride.spec.limitCPUMemoryPercent

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 before 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, you can enable overrides on a per-project basis by applying the following label to the **Namespace** object for each project where you want the overrides to apply. For example, you can configure override so that infrastructure components are not subject to the overrides.

```
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.

For example, a pod has the following resources limits:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  namespace: my-namespace
# ...
spec:
  containers:
    - name: hello-openshift
      image: openshift/hello-openshift
      resources:
        limits:
          memory: "512Mi"
          cpu: "2000m"
# ...
```

The Cluster Resource Override Operator intercepts the original pod request, then overrides the resources according to the configuration set in the **ClusterResourceOverride** object.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  namespace: my-namespace
# ...
spec:
  containers:
    - image: openshift/hello-openshift
      name: hello-openshift
      resources:
        limits:
          cpu: "1"
          memory: 512Mi
        requests:
          cpu: 250m
          memory: 256Mi
# ...
```

where:

spec.containers.resources.limits.cpu

Specifies that the CPU limit has been overridden to **1** because the **limitCPUToMemoryPercent** parameter is set to **200** in the **ClusterResourceOverride** object. As such, 200% of the memory limit, 512Mi in CPU terms, is 1 CPU core.

spec.containers.resources.memory.cpu

Specifies that the CPU request is now **250m** because the **cpuRequestToLimit** is set to **25** in the **ClusterResourceOverride** object. As such, 25% of the 1 CPU core is 250m.

5.9.1. 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 you control overcommit in your cluster.

By default, the installation process creates a Cluster Resource Override Operator pod on a worker node in the **clusterresourceoverride-operator** namespace. You can move this pod to another node, such as an infrastructure node, as needed. Infrastructure nodes are not counted toward the total number of subscriptions that are required to run the environment. For more information, see "Moving the Cluster Resource Override Operator pods".

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

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 **Ecosystem → Software Catalog**.
 - a. Choose **ClusterResourceOverride Operator** from the list of available Operators and click **Install**.
 - b. On the **Install Operator** page, make sure **A specific Namespace on the cluster** is selected for **Installation Mode**.
 - c. Make sure **clusterresourceoverride-operator** is selected for **Installed Namespace**.
 - d. Select an **Update Channel** and **Approval Strategy**.
 - e. Click **Install**.
3. On the **Installed Operators** page, click **ClusterResourceOverride**.
 - a. On the **ClusterResourceOverride Operator** details page, click **Create ClusterResourceOverride**.
 - b. On the **Create ClusterResourceOverride** page, click **YAML view** and edit the YAML template to set the overcommit values as needed:

■

```

apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50
      cpuRequestToLimitPercent: 25
      limitCPUToMemoryPercent: 200

```

where:

metadata.name

Specifies a name for the CR. The name must be **cluster**.

spec.podResourceOverride.spec.memoryRequestToLimitPercent

Specifies the percentage to override the container memory limit, if used, between 1-100. The default is **50**. This parameter is optional.

spec.podResourceOverride.spec.cpuRequestToLimitPercent

Specifies the percentage to override the container CPU limit, if used, between 1-100. The default is **25**. This parameter is optional.

spec.podResourceOverride.spec.limitCPUToMemoryPercent

Specifies the percentage to override the container memory limit, if used. Scaling 1 Gi of RAM at 100 percent is equal to 1 CPU core. This is processed before overriding the CPU request, if configured. The default is **200**. This parameter is optional.

- c. Click **Create**.
4. Check the current state of the admission webhook by checking the status of the cluster custom resource:
 - a. On the **ClusterResourceOverride Operator** page, click **cluster**.
 - b. On the **ClusterResourceOverride Details** page, click **YAML**. The **mutatingWebhookConfigurationRef** section displays when the webhook is called.

```

apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride","met
adata":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":{"spec":
{"cpuRequestToLimitPercent":25,"limitCPUToMemoryPercent":200,"memoryRequestToLi
mitPercent":50}}}}
  creationTimestamp: "2019-12-18T22:35:02Z"
  generation: 1
  name: cluster
  resourceVersion: "127622"
  selfLink: /apis/operator.autoscaling.openshift.io/v1/clusterresourceoverrides/cluster
  uid: 978fc959-1717-4bd1-97d0-ae00ee111e8d
spec:

```

```

    podResourceOverride:
      spec:
        cpuRequestToLimitPercent: 25
        limitCPUToMemoryPercent: 200
        memoryRequestToLimitPercent: 50
      status:

# ...

    mutatingWebhookConfigurationRef:
      apiVersion: admissionregistration.k8s.io/v1
      kind: MutatingWebhookConfiguration
      name: clusterresourceoverrides.admission.autoscaling.openshift.io
      resourceVersion: "127621"
      uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

# ...

```

where:

status.mutatingWebhookConfigurationRef

Specifies the **ClusterResourceOverride** admission webhook.

5.9.2. Installing the Cluster Resource Override Operator using the CLI

You can use the OpenShift CLI to install the Cluster Resource Override Operator to help you control overcommit in your cluster.

By default, the installation process creates a Cluster Resource Override Operator pod on a worker node in the **clusterresourceoverride-operator** namespace. You can move this pod to another node, such as an infrastructure node, as needed. Infrastructure nodes are not counted toward the total number of subscriptions that are required to run the environment. For more information, see "Moving the Cluster Resource Override Operator pods".

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

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-namespaces.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: "stable"
  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
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50
      cpuRequestToLimitPercent: 25
      limitCPUToMemoryPercent: 200
```

where

metadata.name

Specifies a name for the CR. The name must be **cluster**.

spec.podResourceOverride.spec.memoryRequestToLimitPercent

Specifies the percentage to override the container memory limit, if used, between 1-100. The default is **50**. This parameter is optional.

spec.podResourceOverride.spec.cpuRequestToLimitPercent

Specifies the percentage to override the container CPU limit, if used, between 1-100. The default is **25**. This parameter is optional.

spec.podResourceOverride.spec.limitCPUToMemoryPercent

Specifies the percentage to override the container memory limit, if used. Scaling 1 Gi of RAM at 100 percent is equal to 1 CPU core. This is processed before overriding the CPU request, if configured. The default is **200**. This parameter is optional.

- 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 displays 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:
  apiVersion: admissionregistration.k8s.io/v1
  kind: MutatingWebhookConfiguration
  name: clusterresourceoverrides.admission.autoscaling.openshift.io
  resourceVersion: "127621"
  uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

# ...

```

where:

status.mutatingWebhookConfigurationRef

Specifies the **ClusterResourceOverride** admission webhook.

5.9.3. Configuring cluster-level overcommit

You can use the OpenShift CLI to configure the Cluster Resource Override Operator to help control overcommit in your cluster.

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.

By default, the installation process creates two Cluster Resource Override pods on the control plane nodes in the **clusterresourceoverride-operator** namespace. You can move these pods to other nodes, such as infrastructure nodes, as needed. Infrastructure nodes are not counted toward the total number of subscriptions that are required to run the environment. For more information, see "Moving the Cluster Resource Override Operator pods".

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

1. Edit the **ClusterResourceOverride** CR:

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50
      cpuRequestToLimitPercent: 25
      limitCPUToMemoryPercent: 200
# ...
```

where:

spec.podResourceOverride.spec.memoryRequestToLimitPercent

Specifies the percentage to override the container memory limit, if used, between 1-100. The default is **50**. This parameter is optional.

spec.podResourceOverride.spec.cpuRequestToLimitPercent

Specifies the percentage to override the container CPU limit, if used, between 1-100. The default is **25**. This parameter is optional.

spec.podResourceOverride.spec.limitCPUToMemoryPercent

Specifies 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 before overriding the CPU request, if configured. The default is **200**. This parameter is optional.

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"
# ...
```

where:

metadata.labels.clusterresourceoverrides.admission.autoscaling.openshift.io/enabled:
"true"

Specifies that you want to use the Cluster Resource Override Operator with this project.

5.10. 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.

5.10.1. Understanding container CPU and memory requests

Review the following information to learn about container CPU and memory requests to help you ensure that your cluster is properly configured.

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, 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.

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.

5.10.2. Understanding overcommitment and quality of service classes

You can use Quality of Service (QoS) classes in an overcommitted environment to help you ensure that your cluster is properly configured.

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, the pods on the node might attempt to use more compute resource than is available at any given point in time. When this occurs, the node must give priority to one pod over another. The facility used to make this decision is referred to as a Quality of Service (QoS) class.

A pod is designated as one of three QoS classes with decreasing order of priority:

Table 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 pod is classified as Guaranteed .
2	Burstable	If requests and optionally limits are set (not equal to 0) for all resources, and they are not equal, then the pod is classified as Burstable .

Priority	Class Name	Description
3 (lowest)	BestEffort	If requests and limits are not set for any of the resources, then the pod is classified as BestEffort .

Memory is an incompressible resource, so in low memory situations, containers that have the lowest priority are terminated first:

- **Guaranteed** containers are considered top priority, and are guaranteed to only be terminated if they exceed their limits, or if the system is under memory pressure and there are no lower priority containers that can be evicted.
- **Burstable** containers under system memory pressure are more likely to be terminated when 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.

5.10.2.1. 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%** prevents 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%** allows 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%** allows 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 does not have access to requested memory. This condition effectively disables this feature.

5.10.3. Understanding swap memory and QoS

Review the following information to learn how swap memory and QoS interact in an overcommitted environment to help you ensure that your cluster is properly configured.

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 causes nodes to not recognize 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. Out-of-resource handling allows pods to be evicted from a node when it is under memory pressure, and rescheduled on an alternative node that has no such pressure.

5.10.4. Understanding nodes overcommitment

To maintain optimal system performance and stability in an overcommitted environment in OpenShift Container Platform, configure your nodes to manage resource contention effectively.

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 to not 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 the OOM killer in an Out of Memory (OOM) condition, which kills processes based on priority.

You can view the current setting by running the following commands on your nodes:

```
$ sysctl -a |grep commit
```

Example output

```
#...
vm.overcommit_memory = 0
#...
```

```
$ sysctl -a |grep panic
```

Example output

```
#...
vm.panic_on_oom = 0
#...
```



NOTE

The previous commands should already be set on nodes, so 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

Additional resources

- [Disabling or enforcing CPU limits using CPU CFS quotas](#)
- [Reserving resources for system processes](#)
- [Understanding how to reserve memory across quality of service tiers](#)

5.10.5. Disabling or enforcing CPU limits using CPU CFS quotas

You can disable the default enforcement of CPU limits for nodes in a machine config pool.

By default, nodes 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

- You have the label associated with the static **MachineConfigPool** CRD for the type of node you want to configure.

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
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ""
  kubeletConfig:
    cpuCfsQuota: false
```

where:

metadata.name

Specifies a name for the CR.

spec.machineConfigPoolSelector.matchLabels

Specifies the label from the machine config pool.

spec.kubeletConfig.cpuCfsQuota

Specifies the **cpuCfsQuota** parameter to **false**.

2. Run the following command to create the CR:

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

5.10.6. Reserving resources for system processes

You can explicitly reserve resources for non-pod processes by allocating node resources through specifying resources available for scheduling.

To provide more reliable scheduling and minimize node resource overcommitment, each node can reserve a portion of its resources for use by the system daemons that are required to run on your node for your cluster to function.

**NOTE**

It is recommended that you reserve resources for incompressible resources such as memory.

For more details, see *Allocating Resources for Nodes* in the *Additional resources* section.

Additional resources

- [Allocating resources for nodes](#)

5.10.7. Disabling overcommitment for a node

When overcommitment is enabled on a node, you can disable overcommitment on that node. Disabling overcommit can help ensure predictability, stability, and high performance in your cluster.

Procedure

- Run the following command on a node to disable overcommitment on that node:

```
$ sysctl -w vm.overcommit_memory=0
```

5.11. 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 the *Additional resources* section.

Alternatively, you can disable overcommitment for specific projects.

5.11.1. Disabling overcommitment for a project

If overcommitment is enabled on a project, you can disable overcommitment for that projects. This allows infrastructure components to be configured independently of overcommitment.

Procedure

1. Create or edit the namespace object file.
2. Add the following annotation:

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    quota.openshift.io/cluster-resource-override-enabled: "false"
# ...
```

where:

metadata.annotations.quota.openshift.io/cluster-resource-override-enabled.false

Specifies that overcommit is disabled for this namespace.

5.12. FREEING NODE RESOURCES USING GARBAGE COLLECTION

Understand and use garbage collection.

5.12.1. Understanding how terminated containers are removed through garbage collection

Container garbage collection removes terminated containers by using eviction thresholds.

When eviction thresholds are set for garbage collection, the node tries to keep any container for any pod accessible from the API. If the pod has been deleted, the containers will be as well. Containers are preserved as long the pod is not deleted and the eviction threshold is not reached. If the node is under disk pressure, it will remove containers and their logs will no longer be accessible using **oc logs**.

- **eviction-soft** – A soft eviction threshold pairs an eviction threshold with a required administrator-specified grace period.
- **eviction-hard** – A hard eviction threshold has no grace period, and if observed, OpenShift Container Platform takes immediate action.

The following table lists the eviction thresholds:

Table 5.3. Variables for configuring container garbage collection

Node condition	Eviction signal	Description
MemoryPressure	memory.available	The available memory on the node.

Node condition	Eviction signal	Description
DiskPressure	nodefs.available nodefs.inodesFree imagefs.available imagefs.inodesFree	The available disk space or inodes on the node root file system, nodefs , or image file system, imagefs .

**NOTE**

For **evictionHard** you must specify all of these parameters. If you do not specify all parameters, only the specified parameters are applied and the garbage collection will not function properly.

If a node is oscillating above and below a soft eviction threshold, but not exceeding its associated grace period, the corresponding node would constantly oscillate between **true** and **false**. As a consequence, the scheduler could make poor scheduling decisions.

To protect against this oscillation, use the **evictionpressure-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.

**NOTE**

Setting the **evictionPressureTransitionPeriod** parameter to **0** configures the default value of 5 minutes. You cannot set an eviction pressure transition period to zero seconds.

5.12.2. Understanding how images are removed through garbage collection

Image garbage collection removes images that are not referenced by any running pods.

OpenShift Container Platform determines which images to remove from a node based on the disk usage that is reported by **cAdvisor**.

The policy for image garbage collection is based on two conditions:

- The percent of disk usage (expressed as an integer) which triggers image garbage collection. The default is **85**.
- The percent of disk usage (expressed as an integer) to which image garbage collection attempts to free. Default is **80**.

For image garbage collection, you can modify any of the following variables using a custom resource.

Table 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 . This value must be greater than the imageGCLowThresholdPercent value.
imageGCLowThresholdPercent	The percent of disk usage, expressed as an integer, to which image garbage collection attempts to free. The default is 80 . This value must be less than the imageGCHighThresholdPercent value.

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.

5.12.3. Configuring garbage collection for containers and images

As an administrator, you can configure how OpenShift Container Platform performs garbage collection by creating a **kubeletConfig** object for each machine config pool.



NOTE

OpenShift Container Platform supports only one **kubeletConfig** object for each machine config pool.

You can configure any combination of the following:

- Soft eviction for containers
- Hard eviction for containers
- Eviction for images

Container garbage collection removes terminated containers. Image garbage collection removes images that are not referenced by any running pods.

Prerequisites

1. Obtain the label associated with the static **MachineConfigPool** CRD for the type of node you want to configure by entering the following command:

```
$ oc edit machineconfigpool <name>
```

For example:

```
$ oc edit machineconfigpool worker
```

Example output

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" 1
  name: worker
#...
```

1 The label appears under Labels.

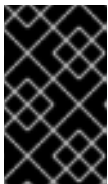
TIP

If the label is not present, add a key/value pair such as:

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

Procedure

1. Create a custom resource (CR) for your configuration change.



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:
      pools.operator.machineconfiguration.openshift.io/worker: "" 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: 6
memory.available: "200Mi"
nodefs.available: "5%"
nodefs.inodesFree: "4%"
imagefs.available: "10%"
imagefs.inodesFree: "5%"
evictionPressureTransitionPeriod: 3m 7
imageMinimumGCAge: 5m 8
imageGCHighThresholdPercent: 80 9
imageGCLowThresholdPercent: 75 10
#...
```

- 1 Name for the object.
- 2 Specify the label from the machine config pool.
- 3 For container garbage collection: Type of eviction: **evictionSoft** or **evictionHard**.
- 4 For container garbage collection: Eviction thresholds based on a specific eviction trigger signal.
- 5 For container garbage collection: Grace periods for the soft eviction. This parameter does not apply to **eviction-hard**.
- 6 For container garbage collection: Eviction thresholds based on a specific eviction trigger signal. For **evictionHard** you must specify all of these parameters. If you do not specify all parameters, only the specified parameters are applied and the garbage collection will not function properly.
- 7 For container garbage collection: The duration to wait before transitioning out of an eviction pressure condition. Setting the **evictionPressureTransitionPeriod** parameter to **0** configures the default value of 5 minutes.
- 8 For image garbage collection: The minimum age for an unused image before the image is removed by garbage collection.
- 9 For image garbage collection: Image garbage collection is triggered at the specified percent of disk usage (expressed as an integer). This value must be greater than the **imageGCLowThresholdPercent** value.
- 10 For image garbage collection: Image garbage collection attempts to free resources to the specified percent of disk usage (expressed as an integer). This value must be less than the **imageGCHighThresholdPercent** value.

2. Run the following command to create the CR:

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

For example:

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

Example output

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

Verification

1. Verify that garbage collection is active by entering the following command. The Machine Config Pool you specified in the custom resource appears with **UPDATING** as 'true' until the change is fully implemented:

```
$ oc get machineconfigpool
```

Example output

NAME	CONFIG	UPDATED	UPDATING
master	rendered-master-546383f80705bd5aeaba93	True	False
worker	rendered-worker-b4c51bb33ccaae6fc4a6a5	False	True

5.13. USING THE NODE TUNING OPERATOR

Understand and use the Node Tuning Operator.

The Node Tuning Operator helps you manage node-level tuning by orchestrating the TuneD daemon and achieves low latency performance by using the Performance Profile controller. The majority of high-performance applications require some level of kernel tuning. The Node Tuning Operator provides a unified management interface to users of node-level sysctls and more flexibility to add custom tuning specified by user needs.

The Operator manages the containerized TuneD daemon for OpenShift Container Platform as a Kubernetes daemon set. It ensures the custom tuning specification is passed to all containerized TuneD daemons running in the cluster in the format that the daemons understand. The daemons run on all nodes in the cluster, one per node.

Node-level settings applied by the containerized TuneD daemon are rolled back on an event that triggers a profile change or when the containerized TuneD daemon is terminated gracefully by receiving and handling a termination signal.

The Node Tuning Operator uses the Performance Profile controller to implement automatic tuning to achieve low latency performance for OpenShift Container Platform applications.

The cluster administrator configures a performance profile to define node-level settings such as the following:

- Updating the kernel to kernel-rt.
- Choosing CPUs for housekeeping.

- Choosing CPUs for running workloads.

The Node Tuning Operator is part of a standard OpenShift Container Platform installation in version 4.1 and later.



NOTE

In earlier versions of OpenShift Container Platform, the Performance Addon Operator was used to implement automatic tuning to achieve low latency performance for OpenShift applications. In OpenShift Container Platform 4.11 and later, this functionality is part of the Node Tuning Operator.

5.13.1. Accessing an example Node Tuning Operator specification

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

Procedure

- Run the following command to access an example Node Tuning Operator specification:

```
oc get tuned.tuned.openshift.io/default -o yaml -n openshift-cluster-node-tuning-operator
```

The default CR is meant for delivering standard node-level tuning for the OpenShift Container Platform platform and it can only be modified to set the Operator Management state. Any other custom changes to the default CR will be overwritten by the Operator. For custom tuning, create your own Tuned CRs. Newly created CRs will be combined with the default CR and custom tuning applied to OpenShift Container Platform nodes based on node or pod labels and profile priorities.



WARNING

While in certain situations the support for pod labels can be a convenient way of automatically delivering required tuning, this practice is discouraged and strongly advised against, especially in large-scale clusters. The default Tuned CR ships without pod label matching. If a custom profile is created with pod label matching, then the functionality will be enabled at that time. The pod label functionality will be deprecated in future versions of the Node Tuning Operator.

5.13.2. Custom tuning specification

The custom resource (CR) for the Operator has two major sections. The first section, **profile:**, is a list of Tuned profiles and their names. The second, **recommend:**, defines the profile selection logic.

Multiple custom tuning specifications can co-exist as multiple CRs in the Operator's namespace. The existence of new CRs or the deletion of old CRs is detected by the Operator. All existing custom tuning specifications are merged and appropriate objects for the containerized Tuned daemons are updated.

Management state

The Operator Management state is set by adjusting the default Tuned CR. By default, the Operator is in the Managed state and the **spec.managementState** field is not present in the default Tuned CR. Valid values for the Operator Management state are as follows:

- Managed: the Operator will update its operands as configuration resources are updated
- Unmanaged: the Operator will ignore changes to the configuration resources
- Removed: the Operator will remove its operands and resources the Operator provisioned

Profile data

The **profile:** section lists TuneD profiles and their names.

```
profile:
- name: tuned_profile_1
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_1 profile

    [sysctl]
    net.ipv4.ip_forward=1
    # ... other sysctl's or other TuneD daemon plugins supported by the containerized TuneD

# ...

- name: tuned_profile_n
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_n profile

    # tuned_profile_n profile settings
```

Recommended profiles

The **profile:** selection logic is defined by the **recommend:** section of the CR. The **recommend:** section is a list of items to recommend the profiles based on a selection criteria.

```
recommend:
<recommend-item-1>
# ...
<recommend-item-n>
```

The individual items of the list:

```
- machineConfigLabels: ❶
  <mcLabels> ❷
  match: ❸
  <match> ❹
  priority: <priority> ❺
  profile: <tuned_profile_name> ❻
  operand: ❼
```



```

debug: <bool> 8
tunedConfig:
  reapply_sysctl: <bool> 9

```

- 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**.
- 7 Optional operand configuration.
- 8 Turn debugging on or off for the Tuned daemon. Options are **true** for on or **false** for off. The default is **false**.
- 9 Turn **reapply_sysctl** functionality on or off for the Tuned daemon. Options are **true** for on and **false** for off.

<match> is an optional list recursively defined as follows:

```

- label: <label_name> 1
  value: <label_value> 2
  type: <label_type> 3
  <match> 4

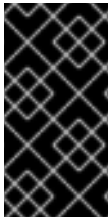
```

- 1 Node or pod label name.
- 2 Optional node or pod label value. If omitted, the presence of **<label_name>** is enough to match.
- 3 Optional object type (**node** or **pod**). If omitted, **node** is assumed.
- 4 An optional **<match>** list.

If **<match>** is not omitted, all nested **<match>** sections must also evaluate to **true**. Otherwise, **false** is assumed and the profile with the respective **<match>** section will not be applied or recommended. Therefore, the nesting (child **<match>** sections) works as logical AND operator. Conversely, if any item of the **<match>** list matches, the entire **<match>** list evaluates to **true**. Therefore, the list acts as logical OR operator.

If **machineConfigLabels** is defined, machine config pool based matching is turned on for the given **recommend:** list item. **<mcLabels>** specifies the labels for a machine config. The machine config is created automatically to apply host settings, such as kernel boot parameters, for the profile **<tuned_profile_name>**. This involves finding all machine config pools with machine config selector matching **<mcLabels>** and setting the profile **<tuned_profile_name>** on all nodes that are assigned the found machine config pools. To target nodes that have both master and worker roles, you must use the master role.

The list items **match** and **machineConfigLabels** are connected by the logical OR operator. The **match** item is evaluated first in a short-circuit manner. Therefore, if it evaluates to **true**, the **machineConfigLabels** item is not considered.



IMPORTANT

When using machine config pool based matching, it is advised to group nodes with the same hardware configuration into the same machine config pool. Not following this practice might result in TuneD operands calculating conflicting kernel parameters for two or more nodes sharing the same machine config pool.

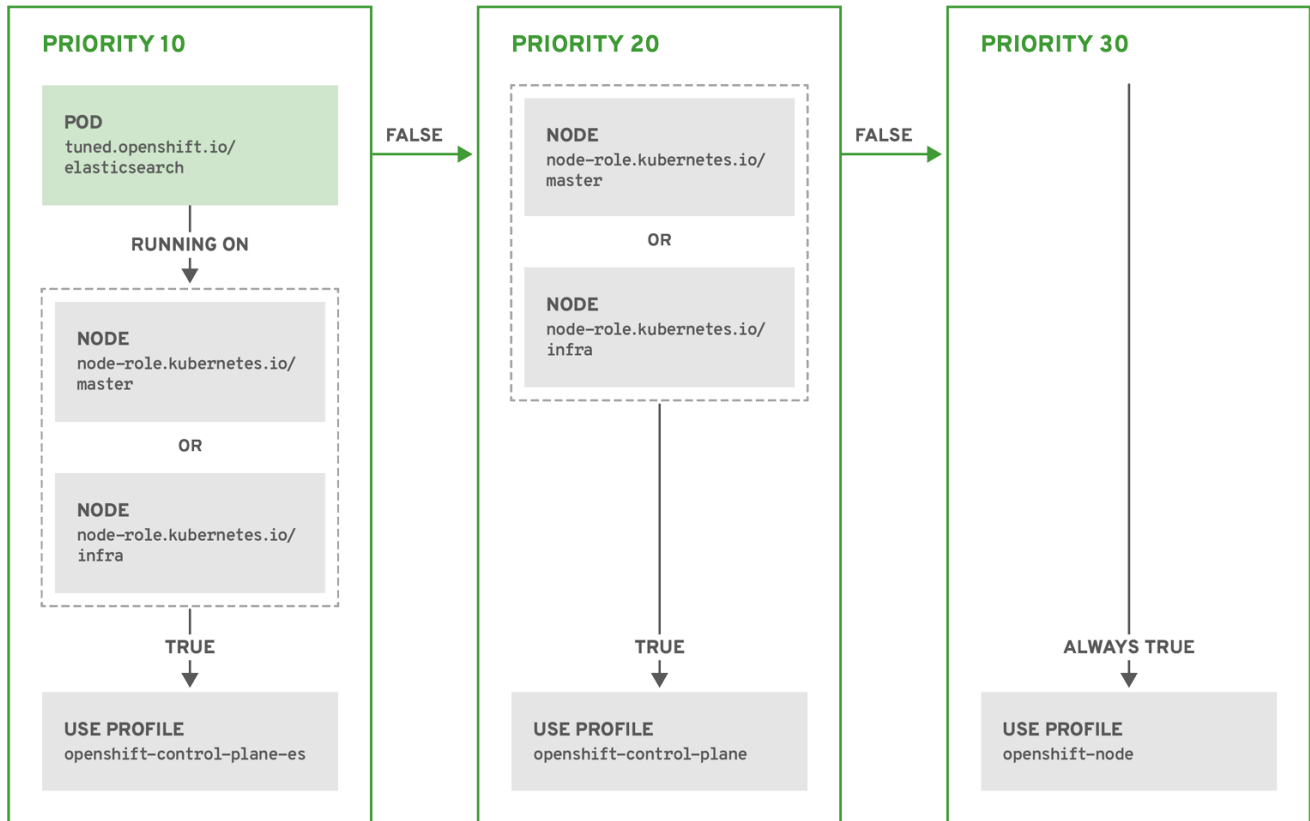
Example: Node or pod label based matching

```
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
      - label: node-role.kubernetes.io/master
      - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra
  priority: 20
  profile: openshift-control-plane
- priority: 30
  profile: openshift-node
```

The CR above is translated for the containerized TuneD daemon into its **recommend.conf** file based on the profile priorities. The profile with the highest priority (**10**) is **openshift-control-plane-es** and, therefore, it is considered first. The containerized TuneD daemon running on a given node looks to see if there is a pod running on the same node with the **tuned.openshift.io/elasticsearch** label set. If not, the entire **<match>** section evaluates as **false**. If there is such a pod with the label, in order for the **<match>** section to evaluate to **true**, the node label also needs to be **node-role.kubernetes.io/master** or **node-role.kubernetes.io/infra**.

If the labels for the profile with priority **10** matched, **openshift-control-plane-es** profile is applied and no other profile is considered. If the node/pod label combination did not match, the second highest priority profile (**openshift-control-plane**) is considered. This profile is applied if the containerized TuneD pod runs on a node with labels **node-role.kubernetes.io/master** or **node-role.kubernetes.io/infra**.

Finally, the profile **openshift-node** has the lowest priority of **30**. It lacks the **<match>** section and, therefore, will always match. It acts as a profile catch-all to set **openshift-node** profile, if no other profile with higher priority matches on a given node.



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.

Cloud provider-specific Tuned profiles

With this functionality, all Cloud provider-specific nodes can conveniently be assigned a TuneD profile specifically tailored to a given Cloud provider on a OpenShift Container Platform cluster. This can be accomplished without adding additional node labels or grouping nodes into machine config pools.

This functionality takes advantage of **spec.providerID** node object values in the form of **<cloud-provider>://<cloud-provider-specific-id>** and writes the file **/var/lib/ocp-tuned/provider** with the value **<cloud-provider>** in NTO operand containers. The content of this file is then used by TuneD to load **provider-<cloud-provider>** profile if such profile exists.

The **openshift** profile that both **openshift-control-plane** and **openshift-node** profiles inherit settings from is now updated to use this functionality through the use of conditional profile loading. Neither NTO nor TuneD currently include any Cloud provider-specific profiles. However, it is possible to create a custom profile **provider-<cloud-provider>** that will be applied to all Cloud provider-specific cluster nodes.

Example GCE Cloud provider profile

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: provider-gce
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
    - data: |
        [main]
        summary=GCE Cloud provider-specific profile
        # Your tuning for GCE Cloud provider goes here.
      name: provider-gce
```



NOTE

Due to profile inheritance, any setting specified in the **provider-<cloud-provider>** profile will be overwritten by the **openshift** profile and its child profiles.

5.13.3. Default profiles set on a cluster

The following are the default profiles set on a cluster.

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: default
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
    - data: |
        [main]
        summary=Optimize systems running OpenShift (provider specific parent profile)
        include=-provider-${f:exec:cat:/var/lib/ocp-tuned/provider},openshift
      name: openshift
  recommend:
    - profile: openshift-control-plane
  priority: 30
```

```

match:
- label: node-role.kubernetes.io/master
- label: node-role.kubernetes.io/infra
- profile: openshift-node
priority: 40

```

Starting with OpenShift Container Platform 4.9, all OpenShift TuneD profiles are shipped with the TuneD package. You can use the **oc exec** command to view the contents of these profiles:

```

$ oc exec $tuned_pod -n openshift-cluster-node-tuning-operator -- find /usr/lib/tuned/openshift{-
control-plane,-node} -name tuned.conf -exec grep -H ^ {} \;

```

5.13.4. Supported TuneD daemon plugins

Excluding the **[main]** section, the following TuneD plugins are supported when using custom profiles defined in the **profile:** section of the Tuned CR:

- audio
- cpu
- disk
- eeepc_she
- modules
- mounts
- net
- scheduler
- scsi_host
- selinux
- sysctl
- sysfs
- usb
- video
- vm
- bootloader

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

- script
- systemd

**NOTE**

The TuneD bootloader plugin only supports Red Hat Enterprise Linux CoreOS (RHCOS) worker nodes.

Additional resources

- [Available TuneD Plugins](#)
- [Getting Started with TuneD](#)

5.14. CONFIGURING THE MAXIMUM NUMBER OF PODS PER NODE

Two parameters control the maximum number of pods that can be scheduled to a node: **podsPerCore** and **maxPods**. If you use both options, the lower of the two limits the number of pods on a node.

For example, if **podsPerCore** is set to **10** on a node with 4 processor cores, the maximum number of pods allowed on the node will be 40.

Prerequisites

1. Obtain the label associated with the static **MachineConfigPool** CRD for the type of node you want to configure by entering the following command:

```
$ oc edit machineconfigpool <name>
```

For example:

```
$ oc edit machineconfigpool worker
```

Example output

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" 1
  name: worker
#...
```

- 1 The label appears under Labels.

TIP

If the label is not present, add a key/value pair such as:

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

Procedure

1. Create a custom resource (CR) for your configuration change.

Sample configuration for a **max-pods** CR

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods ❶
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" ❷
  kubeletConfig:
    podsPerCore: 10 ❸
    maxPods: 250 ❹
#...
```

- ❶ Assign a name to CR.
- ❷ Specify the label from the machine config pool.
- ❸ Specify the number of pods the node can run based on the number of processor cores on the node.
- ❹ Specify the number of pods the node can run to a fixed value, regardless of the properties of the node.



NOTE

Setting **podsPerCore** to **0** disables this limit.

In the above example, the default value for **podsPerCore** is **10** and the default value for **maxPods** is **250**. This means that unless the node has 25 cores or more, by default, **podsPerCore** will be the limiting factor.

2. Run the following command to create the CR:

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

Verification

1. List the **MachineConfigPool** CRDs to see if the change is applied. The **UPDATING** column reports **True** if the change is picked up by the Machine Config Controller:

```
$ oc get machineconfigpools
```

Example output

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
master	master-9cc2c72f205e103bb534	False	False	False
worker	worker-8cecd1236b33ee3f8a5e	False	True	False

Once the change is complete, the **UPDATED** column reports **True**.

```
$ oc get machineconfigpools
```

Example output

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
master	master-9cc2c72f205e103bb534	False	True	False
worker	worker-8cecd1236b33ee3f8a5e	True	False	False

5.15. MACHINE SCALING WITH STATIC IP ADDRESSES

After you deployed your cluster to run nodes with static IP addresses, you can scale an instance of a machine or a machine set to use one of these static IP addresses.

Additional resources

- [Static IP addresses for vSphere nodes](#)

5.15.1. Scaling machines to use static IP addresses

You can scale additional machine sets to use pre-defined static IP addresses on your cluster. For this configuration, you need to create a machine resource YAML file and then define static IP addresses in this file.

Prerequisites

- You deployed a cluster that runs at least one node with a configured static IP address.

Procedure

1. Create a machine resource YAML file and define static IP address network information in the **network** parameter.

Example of a machine resource YAML file with static IP address information defined in the **network parameter.**

```
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  creationTimestamp: null
  labels:
    machine.openshift.io/cluster-api-cluster: <infrastructure_id>
    machine.openshift.io/cluster-api-machine-role: <role>
    machine.openshift.io/cluster-api-machine-type: <role>
    machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
  name: <infrastructure_id>-<role>
  namespace: openshift-machine-api
spec:
  lifecycleHooks: {}
  metadata: {}
  providerSpec:
    value:
```



```

apiVersion: machine.openshift.io/v1beta1
credentialsSecret:
  name: vsphere-cloud-credentials
diskGiB: 120
kind: VSphereMachineProviderSpec
memoryMiB: 8192
metadata:
  creationTimestamp: null
network:
  devices:
    - gateway: 192.168.204.1 1
      ipAddrs:
        - 192.168.204.8/24 2
      nameservers: 3
        - 192.168.204.1
      networkName: qe-segment-204
numCPUs: 4
numCoresPerSocket: 2
snapshot: ""
template: <vm_template_name>
userDataSecret:
  name: worker-user-data
workspace:
  datacenter: <vcenter_data_center_name>
  datastore: <vcenter_datastore_name>
  folder: <vcenter_vm_folder_path>
  resourcepool: <vsphere_resource_pool>
  server: <vcenter_server_ip>
status: {}

```

- 1** The IP address for the default gateway for the network interface.
- 2** Lists IPv4, IPv6, or both IP addresses that installation program passes to the network interface. Both IP families must use the same network interface for the default network.
- 3** Lists a DNS nameserver. You can define up to 3 DNS nameservers. Consider defining more than one DNS nameserver to take advantage of DNS resolution if that one DNS nameserver becomes unreachable.

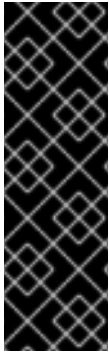
- Create a **machine** custom resource (CR) by entering the following command in your terminal:

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

5.15.2. Machine set scaling of machines with configured static IP addresses

You can use a machine set to scale machines with configured static IP addresses.

After you configure a machine set to request a static IP address for a machine, the machine controller creates an **IPAddressClaim** resource in the **openshift-machine-api** namespace. The external controller then creates an **IPAddress** resource and binds any static IP addresses to the **IPAddressClaim** resource.



IMPORTANT

Your organization might use numerous types of IP address management (IPAM) services. If you want to enable a particular IPAM service on OpenShift Container Platform, you might need to manually create the **IPAddressClaim** resource in a YAML definition and then bind a static IP address to this resource by entering the following command in your **oc** CLI:

```
$ oc create -f <ipaddressclaim_filename>
```

The following demonstrates an example of an **IPAddressClaim** resource:

```
kind: IPAddressClaim
metadata:
  finalizers:
  - machine.openshift.io/ip-claim-protection
  name: cluster-dev-9n5wg-worker-0-m7529-claim-0-0
  namespace: openshift-machine-api
spec:
  poolRef:
    apiGroup: ipamcontroller.example.io
    kind: IPPool
    name: static-ci-pool
status: {}
```

The machine controller updates the machine with a status of **IPAddressClaimed** to indicate that a static IP address has successfully bound to the **IPAddressClaim** resource. The machine controller applies the same status to a machine with multiple **IPAddressClaim** resources that each contain a bound static IP address. The machine controller then creates a virtual machine and applies static IP addresses to any nodes listed in the **providerSpec** of a machine's configuration.

5.15.3. Using a machine set to scale machines with configured static IP addresses

You can use a machine set to scale machines with configured static IP addresses.

The example in the procedure demonstrates the use of controllers for scaling machines in a machine set.

Prerequisites

- You deployed a cluster that runs at least one node with a configured static IP address.

Procedure

1. Configure a machine set by specifying IP pool information in the **network.devices.addressesFromPools** schema of the machine set's YAML file:

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  annotations:
    machine.openshift.io/memoryMb: "8192"
    machine.openshift.io/vCPU: "4"
  labels:
```

```

  machine.openshift.io/cluster-api-cluster: <infrastructure_id>
  name: <infrastructure_id>-<role>
  namespace: openshift-machine-api
spec:
  replicas: 0
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-cluster: <infrastructure_id>
      machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
  template:
    metadata:
      labels:
        ipam: "true"
        machine.openshift.io/cluster-api-cluster: <infrastructure_id>
        machine.openshift.io/cluster-api-machine-role: worker
        machine.openshift.io/cluster-api-machine-type: worker
        machine.openshift.io/cluster-api-machineset: <infrastructure_id>-<role>
    spec:
      lifecycleHooks: {}
      metadata: {}
      providerSpec:
        value:
          apiVersion: machine.openshift.io/v1beta1
          credentialsSecret:
            name: vsphere-cloud-credentials
          diskGiB: 120
          kind: VSphereMachineProviderSpec
          memoryMiB: 8192
          metadata: {}
          network:
            devices:
              - addressesFromPools: 1
              - group: ipamcontroller.example.io
                name: static-ci-pool
                resource: IPPool
              nameservers:
                - "192.168.204.1" 2
              networkName: qe-segment-204
            numCPUs: 4
            numCoresPerSocket: 2
            snapshot: ""
            template: rvanderp4-dev-9n5wg-rhcos-generated-region-generated-zone
            userDataSecret:
              name: worker-user-data
          workspace:
            datacenter: IBMCDatacenter
            datastore: /IBMCDatacenter/datastore/vsanDatastore
            folder: /IBMCDatacenter/vm/rvanderp4-dev-9n5wg
            resourcePool: /IBMCDatacenter/host/IBMCCluster/Resources
            server: vcenter.ibmcd.devcluster.openshift.com

```

- 1 Specifies an IP pool, which lists a static IP address or a range of static IP addresses. The IP Pool can either be a reference to a custom resource definition (CRD) or a resource supported by the **IPAddressClaims** resource handler. The machine controller accesses static IP addresses listed in the machine set's configuration and then allocates each address to each machine.

- 2 Lists a nameserver. You must specify a nameserver for nodes that receive static IP address, because the Dynamic Host Configuration Protocol (DHCP) network configuration

2. Scale the machine set by entering the following commands in your **oc** CLI:

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

Or:

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

After each machine is scaled up, the machine controller creates an **IPAddressClaim** resource.

3. Optional: Check that the **IPAddressClaim** resource exists in the **openshift-machine-api** namespace by entering the following command:

```
$ oc get ipaddressclaims.ipam.cluster.x-k8s.io -n openshift-machine-api
```

Example oc CLI output that lists two IP pools listed in the **openshift-machine-api namespace**

```
NAME                                POOL NAME      POOL KIND
cluster-dev-9n5wg-worker-0-m7529-claim-0-0  static-ci-pool IPPool
cluster-dev-9n5wg-worker-0-wdqkt-claim-0-0  static-ci-pool IPPool
```

4. Create an **IPAddress** resource by entering the following command:

```
$ oc create -f ipaddress.yaml
```

The following example shows an **IPAddress** resource with defined network configuration information and one defined static IP address:

```
apiVersion: ipam.cluster.x-k8s.io/v1alpha1
kind: IPAddress
metadata:
  name: cluster-dev-9n5wg-worker-0-m7529-ipaddress-0-0
  namespace: openshift-machine-api
spec:
  address: 192.168.204.129
  claimRef: 1
    name: cluster-dev-9n5wg-worker-0-m7529-claim-0-0
  gateway: 192.168.204.1
  poolRef: 2
    apiGroup: ipamcontroller.example.io
    kind: IPPool
    name: static-ci-pool
  prefix: 23
```

- 1 The name of the target **IPAddressClaim** resource.
- 2 Details information about the static IP address or addresses from your nodes.

**NOTE**

By default, the external controller automatically scans any resources in the machine set for recognizable address pool types. When the external controller finds **kind: IPPool** defined in the **IPAddress** resource, the controller binds any static IP addresses to the **IPAddressClaim** resource.

5. Update the **IPAddressClaim** status with a reference to the **IPAddress** resource:

```
$ oc --type=merge patch IPAddressClaim cluster-dev-9n5wg-worker-0-m7529-claim-0-0 -  
p='{"status":{"addressRef": {"name": "cluster-dev-9n5wg-worker-0-m7529-ipaddress-0-0"}}}' -  
n openshift-machine-api --subresource=status
```

CHAPTER 6. POSTINSTALLATION NETWORK CONFIGURATION

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

6.1. USING THE CLUSTER NETWORK OPERATOR

You can use the Cluster Network Operator (CNO) to deploy and manage cluster network components on an OpenShift Container Platform cluster, including the Container Network Interface (CNI) network plugin selected for the cluster during installation.

For more information, see [Cluster Network Operator in OpenShift Container Platform](#) .

6.2. NETWORK CONFIGURATION TASKS

- [Configuring the cluster-wide proxy](#)
- [Configuring ingress cluster traffic overview](#)
- [Configuring the node port service range](#)
- [Configuring IPsec encryption](#)
- [Create a network policy or configure multitenant isolation with network policies](#)
- [Optimizing routing](#)
- [Understanding multiple networks](#)

6.2.1. 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.

6.2.1.1. 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:

Prerequisites

- You have access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.

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.

6.2.1.2. 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 container network interface (CNI) network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes.

- 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:
            policy-group.network.openshift.io/ingress:
    podSelector: {}
    policyTypes:
    - Ingress
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-kube-apiserver-operator
  spec:
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: openshift-kube-apiserver-operator
    podSelector:
      matchLabels:
```



```

    app: kube-apiserver-operator
  policyTypes:
  - Ingress
  ...

```

3. Optional: Create a new project and confirm the successful creation of your network policy objects.

- 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
```

Expected output:

NAME	POD-SELECTOR	AGE
allow-from-openshift-ingress	<none>	7s
allow-from-same-namespace	<none>	7s

CHAPTER 7. CONFIGURING IMAGE STREAMS AND IMAGE REGISTRIES

You can update the global pull secret for your cluster by either replacing the current pull secret or appending a new pull secret. The procedure is required when users use a separate registry to store images than the registry used during installation. For more information, see [Using image pull secrets](#).

For information about images and configuring image streams or image registries, see the following documentation:

- [Overview of images](#)
- [Image Registry Operator in OpenShift Container Platform](#)
- [Configuring image registry settings](#)

7.1. 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.

7.1.1. Cluster Samples Operator assistance for mirroring

During installation, OpenShift Container Platform creates a config map named **imagestreamtag-to-image** in the **openshift-cluster-samples-operator** namespace.

The **imagestreamtag-to-image** config map contains an entry, the populating image, for each image stream tag.

The format of the key for each entry in the data field in the config map is **<image_stream_name>_<image_stream_tag_name>**.

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



NOTE

The use of samples in a network-restricted or discontinued environment might require access to services external to your network. Some example services include: Github, Maven Central, npm, RubyGems, PyPi and others. There might be additional steps to take that allow the Cluster Samples Operators objects to reach the services they require.

Use the following principles to determine which images you need to mirror 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.

7.1.2. Using Cluster Samples Operator image streams with alternate or mirrored registries

You can use an alternate or mirror registry to host your images streams instead of using the Red Hat registry.

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.



NOTE

The **cli**, **installer**, **must-gather**, and **tests** image streams, while part of the install payload, are not managed by the Cluster Samples Operator. These are not addressed in this procedure.



IMPORTANT

The Cluster Samples Operator must be set to **Managed** in a disconnected environment. To install the image streams, you must have a mirrored registry.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- Create a pull secret for your mirror registry.

Procedure

1. Access the images of a specific image stream to mirror, for example:

```
$ oc get is <imagestream> -n openshift -o json | jq .spec.tags[].from.name | grep registry.redhat.io
```

2. Mirror images from registry.redhat.io associated with any image streams you need

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

3. Create the image configuration object for the cluster by running the following command:

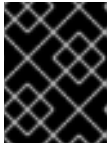
```
$ oc create configmap registry-config --from-file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
```

4. Add the required trusted CAs for the mirror in the image configuration object:

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":{"name":"registry-config"}}}' --type=merge
```

- 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
```



IMPORTANT

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

- 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. You can use **Removed** to purge both the image streams and templates. This eliminates the possibility of attempts to use the templates if they are not functional because of any missing image streams.

7.1.3. 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

- 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:

- Create the cluster's image configuration object:

```
$ oc create configmap registry-config --from-file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
```

- 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
```

- 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)
```

7.2. CONFIGURING PERIODIC IMPORTING OF CLUSTER SAMPLE OPERATOR IMAGE STREAM TAGS

You can ensure that you always have access to the latest versions of the Cluster Sample Operator images by periodically importing the image stream tags when new versions become available.

Procedure

1. Fetch all the imagestreams in the **openshift** namespace by running the following command:

```
oc get imagestreams -n openshift
```

2. Fetch the tags for every imagestream in the **openshift** namespace by running the following command:

```
$ oc get is <image-stream-name> -o jsonpath="{range .spec.tags[*]}{.name}{'\t'}{.from.name}{'\n'}{end}" -n openshift
```

For example:

```
$ oc get is ubi8-openjdk-17 -o jsonpath="{range .spec.tags[*]}{.name}{'\t'}{.from.name}{'\n'}{end}" -n openshift
```

Example output

```
1.11 registry.access.redhat.com/ubi8/openjdk-17:1.11
1.12 registry.access.redhat.com/ubi8/openjdk-17:1.12
```

3. Schedule periodic importing of images for each tag present in the image stream by running the following command:

```
$ oc tag <repository/image> <image-stream-name:tag> --scheduled -n openshift
```

For example:

```
$ oc tag registry.access.redhat.com/ubi8/openjdk-17:1.11 ubi8-openjdk-17:1.11 --scheduled -n openshift
```

```
$ oc tag registry.access.redhat.com/ubi8/openjdk-17:1.12 ubi8-openjdk-17:1.12 --scheduled -n openshift
```



NOTE

Using the **--scheduled** flag is recommended to periodically re-import an image when working with an external container image registry. The **--scheduled** flag helps to ensure that you receive the latest versions and security updates. Additionally, this setting allows the import process to automatically retry if a temporary error initially prevents the image from being imported.

By default, scheduled image imports occur every 15 minutes cluster-wide.

4. Verify the scheduling status of the periodic import by running the following command:

```
oc get imagestream <image-stream-name> -o jsonpath="{range .spec.tags[*]}Tag: {.name} {\t}Scheduled: {.importPolicy.scheduled}{\n}{end}" -n openshift
```

For example:

```
oc get imagestream ubi8-openjdk-17 -o jsonpath="{range .spec.tags[*]}Tag: {.name} {\t}Scheduled: {.importPolicy.scheduled}{\n}{end}" -n openshift
```

Example output

```
Tag: 1.11 Scheduled: true
Tag: 1.12 Scheduled: true
```

CHAPTER 8. POSTINSTALLATION STORAGE CONFIGURATION

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

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

Dynamic provisioning

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

Static provisioning

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

8.1. DYNAMIC PROVISIONING

Dynamic Provisioning allows you to create storage volumes on-demand, eliminating the need for cluster administrators to pre-provision storage. See [Dynamic provisioning](#).

8.2. RECOMMENDED CONFIGURABLE STORAGE TECHNOLOGY

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

Table 8.1. Recommended and configurable storage technology

Storage type	Block	File	Object
ROX ¹	Yes ⁴	Yes ⁴	Yes
RWX ²	No	Yes	Yes
Registry	Configurable	Configurable	Recommended
Scaled registry	Not configurable	Configurable	Recommended
Metrics ³	Recommended	Configurable ⁵	Not configurable
Elasticsearch Logging	Recommended	Configurable ⁶	Not supported ⁶
Loki Logging	Not configurable	Not configurable	Recommended
Apps	Recommended	Recommended	Not configurable ⁷
¹ ReadOnlyMany			
² ReadWriteMany Storage type	Block	File	Object
³ Prometheus is the underlying technology used for metrics. ⁴ This does not apply to physical disk, VM physical disk, VMDK, loopback over NFS, AWS EBS, and Azure Disk. ⁵ For metrics, using file storage with the ReadWriteMany (RWX) access mode is unreliable. If you use file storage, do not configure the RWX access mode on any persistent volume claims (PVCs) that are configured for use with metrics. ⁶ For logging, review the recommended storage solution in Configuring persistent storage for the log store section. Using NFS storage as a persistent volume or through NAS, such as Gluster, can corrupt the data. Hence, NFS is not supported for Elasticsearch storage and LokiStack log store in OpenShift Container Platform Logging. You must use one persistent volume type per log store. ⁷ Object storage is not consumed through OpenShift Container Platform's PVs or PVCs. Apps must integrate with the object storage REST API.			

**NOTE**

A scaled registry is an OpenShift image registry where two or more pod replicas are running.

8.2.1. Specific application storage recommendations



IMPORTANT

Testing shows issues with using the NFS server on Red Hat Enterprise Linux (RHEL) as a 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 in 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.

8.2.1.1. Registry

In a non-scaled/high-availability (HA) OpenShift image registry cluster deployment:

- The storage technology does not have to support RWX access mode.
- The storage technology must ensure read-after-write consistency.
- The preferred storage technology is object storage followed by block storage.
- File storage is not recommended for OpenShift image registry cluster deployment with production workloads.

8.2.1.2. Scaled registry

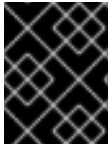
In a scaled/HA OpenShift image registry cluster deployment:

- The storage technology must support RWX access mode.
- The storage technology must ensure read-after-write consistency.
- The preferred storage technology is object storage.
- Red Hat OpenShift Data Foundation, 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.
- The use of Network File System (NFS) storage with OpenShift Container Platform is supported. However, the use of NFS storage with a scaled registry can cause known issues. For more information, see the Red Hat Knowledgebase solution, [Is NFS supported for OpenShift cluster internal components in Production?](#).

8.2.1.3. 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.

8.2.1.4. Logging

In an OpenShift Container Platform hosted logging cluster deployment:

- Loki Operator:
 - The preferred storage technology is S3 compatible Object storage.
 - Block storage is not configurable.
- OpenShift Elasticsearch Operator:
 - The preferred storage technology is block storage.
 - Object storage is not supported.

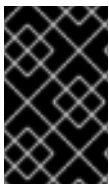
**NOTE**

As of logging version 5.4.3 the OpenShift Elasticsearch Operator is deprecated and is planned to be removed in a future release. Red Hat will provide bug fixes and support for this feature during the current release lifecycle, but this feature will no longer receive enhancements and will be removed. As an alternative to using the OpenShift Elasticsearch Operator to manage the default log storage, you can use the Loki Operator.

8.2.1.5. 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.

8.2.2. Other specific application storage recommendations**IMPORTANT**

It is not recommended to use RAID configurations on **Write** intensive workloads, such as **etcd**. If you are running **etcd** with a RAID configuration, you might be at risk of encountering performance issues with your workloads.

- Red Hat OpenStack Platform (RHOSP) Cinder: RHOSP Cinder tends to be adept in ROX access mode use cases.
- Databases: Databases (RDBMSs, NoSQL DBs, etc.) tend to perform best with dedicated block storage.

- The etcd database must have enough storage and adequate performance capacity to enable a large cluster. Information about monitoring and benchmarking tools to establish ample storage and a high-performance environment is described in *Recommended etcd practices*.

8.3. DEPLOY RED HAT OPENSIFT DATA FOUNDATION

Red Hat OpenShift Data Foundation is a provider of agnostic persistent storage for OpenShift Container Platform supporting file, block, and object storage, either in-house or in hybrid clouds. As a Red Hat storage solution, Red Hat OpenShift Data Foundation is completely integrated with OpenShift Container Platform for deployment, management, and monitoring. For more information, see the [Red Hat OpenShift Data Foundation documentation](#).



IMPORTANT

OpenShift Data Foundation on top of Red Hat Hyperconverged Infrastructure (RHII) for Virtualization, which uses hyperconverged nodes that host virtual machines installed with OpenShift Container Platform, is not a supported configuration. For more information about supported platforms, see the [Red Hat OpenShift Data Foundation Supportability and Interoperability Guide](#).

If you are looking for Red Hat OpenShift Data Foundation information about...	See the following Red Hat OpenShift Data Foundation documentation:
What's new, known issues, notable bug fixes, and Technology Previews	OpenShift Data Foundation 4.12 Release Notes
Supported workloads, layouts, hardware and software requirements, sizing and scaling recommendations	Planning your OpenShift Data Foundation 4.12 deployment
Instructions on deploying OpenShift Data Foundation to use an external Red Hat Ceph Storage cluster	Deploying OpenShift Data Foundation 4.12 in external mode
Instructions on deploying OpenShift Data Foundation to local storage on bare metal infrastructure	Deploying OpenShift Data Foundation 4.12 using bare metal infrastructure
Instructions on deploying OpenShift Data Foundation on Red Hat OpenShift Container Platform VMware vSphere clusters	Deploying OpenShift Data Foundation 4.12 on VMware vSphere
Instructions on deploying OpenShift Data Foundation using Amazon Web Services for local or cloud storage	Deploying OpenShift Data Foundation 4.12 using Amazon Web Services
Instructions on deploying and managing OpenShift Data Foundation on existing Red Hat OpenShift Container Platform Google Cloud clusters	Deploying and managing OpenShift Data Foundation 4.12 using Google Cloud

If you are looking for Red Hat OpenShift Data Foundation information about...	See the following Red Hat OpenShift Data Foundation documentation:
Instructions on deploying and managing OpenShift Data Foundation on existing Red Hat OpenShift Container Platform Azure clusters	Deploying and managing OpenShift Data Foundation 4.12 using Microsoft Azure
Instructions on deploying OpenShift Data Foundation to use local storage on IBM Power® infrastructure	Deploying OpenShift Data Foundation on IBM Power®
Instructions on deploying OpenShift Data Foundation to use local storage on IBM Z® infrastructure	Deploying OpenShift Data Foundation on IBM Z® infrastructure
Allocating storage to core services and hosted applications in Red Hat OpenShift Data Foundation, including snapshot and clone	Managing and allocating resources
Managing storage resources across a hybrid cloud or multicloud environment using the Multicloud Object Gateway (NooBaa)	Managing hybrid and multicloud resources
Safely replacing storage devices for Red Hat OpenShift Data Foundation	Replacing devices
Safely replacing a node in a Red Hat OpenShift Data Foundation cluster	Replacing nodes
Scaling operations in Red Hat OpenShift Data Foundation	Scaling storage
Monitoring a Red Hat OpenShift Data Foundation 4.12 cluster	Monitoring Red Hat OpenShift Data Foundation 4.12
Resolve issues encountered during operations	Troubleshooting OpenShift Data Foundation 4.12
Migrating your OpenShift Container Platform cluster from version 3 to version 4	Migration

CHAPTER 9. 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.

9.1. 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.

9.1.1. 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.

9.1.2. 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](#).

9.1.3. Identity provider parameters

The following parameters are common to all identity providers:

Parameter	Description
name	The provider name is prefixed to provider user names to form an identity name.
mappingMethod	Defines how new identities are mapped to users when they log in. Enter one of the following values: claim The default value. Provisions a user with the identity's preferred user name. Fails if a user with that user name is already mapped to another identity. lookup Looks up an existing identity, user identity mapping, and user, but does not automatically provision users or identities. This allows cluster administrators to set up identities and users manually, or using an external process. Using this method requires you to manually provision users. add Provisions a user with the identity's preferred user name. If a user with that user name already exists, the identity is mapped to the existing user, adding to any existing identity mappings for the user. Required when multiple identity providers are configured that identify the same set of users and map to the same user names.



NOTE

When adding or changing identity providers, you can map identities from the new provider to existing users by setting the **mappingMethod** parameter to **add**.

9.1.4. 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](#).

9.2. USING RBAC TO DEFINE AND APPLY PERMISSIONS

Understand and apply role-based access control.

9.2.1. 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.

9.2.1.1. 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

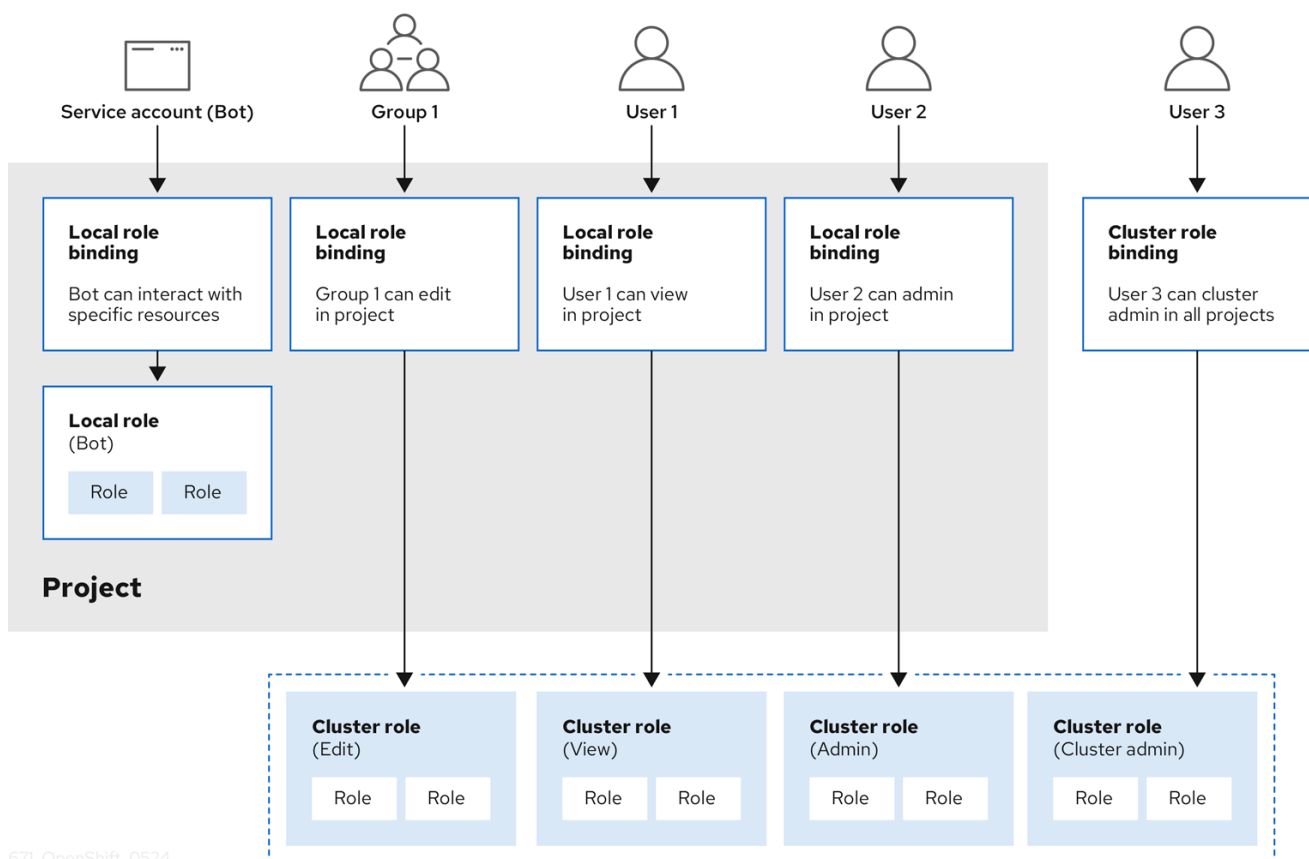
It is not recommended to manually modify the default cluster roles. Modifications to these system roles can prevent a cluster from functioning properly.

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
cluster-reader	A user that can get or view most of the objects but cannot modify them.
edit	A user that can modify most objects in a project but does not have the power to view or modify roles or bindings.
self-provisioner	A user that can create their own projects.
view	A user who cannot make any modifications, but can see most objects in a project. They cannot view or modify roles or bindings.

Be mindful of the difference between local and cluster bindings. For example, if you bind the **cluster-admin** role to a user by using a local role binding, it might appear that this user has the privileges of a cluster administrator. This is not the case. Binding the **cluster-admin** to a user in a project grants super administrator privileges for only that project to the user. That user has the permissions of the cluster role **admin**, plus a few additional permissions like the ability to edit rate limits, for that project. This binding can be confusing via the web console UI, which does not list cluster role bindings that are bound to true cluster administrators. However, it does list local role bindings that you can use to locally bind **cluster-admin**.

The relationships between cluster roles, local roles, cluster role bindings, local role bindings, users, groups and service accounts are illustrated below.



671_OpenShift_0524

**WARNING**

The **get pods/exec**, **get pods/***, and **get *** rules grant execution privileges when they are applied to a role. Apply the principle of least privilege and assign only the minimal RBAC rights required for users and agents. For more information, see [RBAC rules allow execution privileges](#).

9.2.1.2. 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.

9.2.1.2.1. 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.

9.2.2. Projects and namespaces

A Kubernetes *namespace* provides a mechanism to scope resources in a cluster. The [Kubernetes documentation](#) has more information on namespaces.

Namespaces provide a unique scope for:

- Named resources to avoid basic naming collisions.
- Delegated management authority to trusted users.
- The ability to limit community resource consumption.

Most objects in the system are scoped by namespace, but some are excepted and have no namespace, including nodes and users.

A *project* is a Kubernetes namespace with additional annotations and is the central vehicle by which access to resources for regular users is managed. A project allows a community of users to organize and manage their content in isolation from other communities. Users must be given access to projects by administrators, or if allowed to create projects, automatically have access to their own projects.

Projects can have a separate **name**, **displayName**, and **description**.

- The mandatory **name** is a unique identifier for the project and is most visible when using the CLI tools or API. The maximum name length is 63 characters.
- The optional **displayName** is how the project is displayed in the web console (defaults to **name**).
- The optional **description** can be a more detailed description of the project and is also visible in the web console.

Each project scopes its own set of:

Object	Description
Objects	Pods, services, replication controllers, etc.
Policies	Rules for which users can or cannot perform actions on objects.

Object	Description
Constraints	Quotas for each kind of object that can be limited.
Service accounts	Service accounts act automatically with designated access to objects in the project.

Cluster administrators can create projects and delegate administrative rights for the project to any member of the user community. Cluster administrators can also allow developers to create their own projects.

Developers and administrators can interact with projects by using the CLI or the web console.

9.2.3. 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.



IMPORTANT

Do not run workloads in or share access to default projects. Default projects are reserved for running core cluster components.

The following default projects are considered highly privileged: **default**, **kube-public**, **kube-system**, **openshift**, **openshift-infra**, **openshift-node**, and other system-created projects that have the **openshift.io/run-level** label set to **0** or **1**. Functionality that relies on admission plugins, such as pod security admission, security context constraints, cluster resource quotas, and image reference resolution, does not work in highly privileged projects.

9.2.4. 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]			
podd disruptionbudgets.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]

podsecuritypolicy	subjectreviews			[create]
resourceaccess	reviews			[create]
routes/custom-host				[create]
subjectaccess	reviews			[create]
subjectrules	reviews			[create]
deploymentconfig	rollbacks.apps.openshift.io			[create]
deploymentconfigs.apps.openshift.io	instantiate			[create]
deploymentconfigs.apps.openshift.io	rollback			[create]
localsubjectaccess	reviews.authorization.k8s.io			[create]
localresourceaccess	reviews.authorization.openshift.io			[create]
localsubjectaccess	reviews.authorization.openshift.io			[create]
resourceaccess	reviews.authorization.openshift.io			[create]
subjectaccess	reviews.authorization.openshift.io			[create]
subjectrules	reviews.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]
podsecuritypolicy	reviews.security.openshift.io			[create]
podsecuritypolicy	selfsubjectreviews.security.openshift.io			[create]
podsecuritypolicy	subjectreviews.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

.*	[]	[]	[*]
	[*]	[]	[*]

...

- 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
...

```

9.2.5. Viewing local roles and bindings

You can use the **oc** CLI to view local roles and bindings by using the **oc describe** command.

Prerequisites

- Install the **oc** CLI.
- Obtain permission to view the local roles and bindings:
 - Users with the **cluster-admin** default cluster role bound cluster-wide can perform any action on any resource, including viewing local roles and bindings.
 - Users with the **admin** default cluster role bound locally can view and manage roles and bindings in that project.

Procedure

1. To view the current set of local role bindings, which show the users and groups that are bound to various roles for the current project:

```
$ oc describe rolebinding.rbac
```

2. To view the local role bindings for a different project, add the **-n** flag to the command:

```
$ oc describe rolebinding.rbac -n joe-project
```

Example output

```

Name:      admin
Labels:    <none>
Annotations: <none>
Role:
  Kind: ClusterRole
  Name: admin
Subjects:

```

```

Kind  Name      Namespace
----  -
User  kube:admin

Name:      system:deployers
Labels:    <none>
Annotations: openshift.io/description:
            Allows deploymentconfigs in this namespace to rollout pods in
            this namespace. It is auto-managed by a controller; remove
            subjects to disa...

Role:
  Kind: ClusterRole
  Name: system:deployer
Subjects:
  Kind      Name      Namespace
  ----      -
ServiceAccount  deployer  joe-project

Name:      system:image-builders
Labels:    <none>
Annotations: openshift.io/description:
            Allows builds in this namespace to push images to this
            namespace. It is auto-managed by a controller; remove subjects
            to disable.

Role:
  Kind: ClusterRole
  Name: system:image-builder
Subjects:
  Kind      Name      Namespace
  ----      -
ServiceAccount  builder  joe-project

Name:      system:image-pullers
Labels:    <none>
Annotations: openshift.io/description:
            Allows all pods in this namespace to pull images from this
            namespace. It is auto-managed by a controller; remove subjects
            to disable.

Role:
  Kind: ClusterRole
  Name: system:image-puller
Subjects:
  Kind  Name      Namespace
  ----  -
Group  system:serviceaccounts:joe-project

```

9.2.6. 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	

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**.

9.2.7. 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
```

9.2.8. 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
```

9.2.9. 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 9.1. 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.

9.2.10. 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 9.2. Cluster role binding operations

Command	Description
\$ oc adm policy add-cluster-role-to-user <role> <username>	Binds a given role to specified users for all projects in the cluster.
\$ oc adm policy remove-cluster-role-from-user <role> <username>	Removes a given role from specified users for all projects in the cluster.
\$ oc adm policy add-cluster-role-to-group <role> <groupname>	Binds a given role to specified groups for all projects in the cluster.

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

9.2.11. 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>
```

9.2.12. Cluster role bindings for unauthenticated groups



NOTE

Before OpenShift Container Platform 4.17, unauthenticated groups were allowed access to some cluster roles. Clusters updated from versions before OpenShift Container Platform 4.17 retain this access for unauthenticated groups.

For security reasons OpenShift Container Platform 4.20 does not allow unauthenticated groups to have default access to cluster roles.

There are use cases where it might be necessary to add **system:unauthenticated** to a cluster role.

Cluster administrators can add unauthenticated users to the following cluster roles:

- **system:scope-impersonation**
- **system:webhook**
- **system:oauth-token-deleter**
- **self-access-reviewer**



IMPORTANT

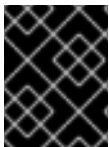
Always verify compliance with your organization's security standards when modifying unauthenticated access.

9.2.13. Adding unauthenticated groups to cluster roles

As a cluster administrator, you can add unauthenticated users to the following cluster roles in OpenShift Container Platform by creating a cluster role binding. Unauthenticated users do not have access to non-public cluster roles. This should only be done in specific use cases when necessary.

You can add unauthenticated users to the following cluster roles:

- **system:scope-impersonation**
- **system:webhook**
- **system:oauth-token-deleter**
- **self-access-reviewer**



IMPORTANT

Always verify compliance with your organization's security standards when modifying unauthenticated access.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a YAML file named **add-<cluster_role>-unauth.yaml** and add the following content:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  name: <cluster_role>access-unauthenticated
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: <cluster_role>
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:unauthenticated
```

2. Apply the configuration by running the following command:

```
$ oc apply -f add-<cluster_role>.yaml
```

9.3. THE KUBEADMIN USER

OpenShift Container Platform creates a cluster administrator, **kubeadmin**, after the installation process completes.

This user has the **cluster-admin** role automatically applied and is treated as the root user for the cluster. The password is dynamically generated and unique to your OpenShift Container Platform environment. After installation completes the password is provided in the installation program's output. For example:

```
INFO Install complete!
INFO Run 'export KUBECONFIG=<your working directory>/auth/kubeconfig' to manage the cluster
with 'oc', the OpenShift CLI.
INFO The cluster is ready when 'oc login -u kubeadmin -p <provided>' succeeds (wait a few minutes).
INFO Access the OpenShift web-console here: https://console-openshift-console.apps.demo1.openshift4-beta-abcorp.com
INFO Login to the console with user: kubeadmin, password: <provided>
```

9.3.1. 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
```

9.4. POPULATING THE SOFTWARE CATALOG FROM MIRRORED OPERATOR CATALOGS

If you mirrored Operator catalogs for use with disconnected clusters, you can populate the software catalog 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.

9.4.1. Prerequisites

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

9.4.1.1. 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.

9.4.1.2. 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. The software catalog uses catalog sources to populate the user interface.

TIP

Alternatively, you can use the web console to manage catalog sources. From the **Administration → Cluster Settings → Configuration → OperatorHub** page, click the **Sources** tab, where you can create, update, delete, disable, and enable individual sources.

Prerequisites

- You built and pushed an index image to a registry.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

- 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.
 - Modify the following to your specifications and save it as a **catalogSource.yaml** file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: my-operator-catalog 1
  namespace: openshift-marketplace 2
spec:
  sourceType: grpc
  grpcPodConfig:
    securityContextConfig: <security_mode> 3
  image: <registry>/<namespace>/redhat-operator-index:v4.20 4
```

```

displayName: My Operator Catalog
publisher: <publisher_name> 5
updateStrategy:
  registryPoll: 6
  interval: 30m

```

- 1 If you mirrored content to local files before uploading to a registry, remove any backslash (/) characters from the **metadata.name** field to avoid an "invalid resource name" error when you create the object.
- 2 If you want the catalog source to be available globally to users in all namespaces, specify the **openshift-marketplace** namespace. Otherwise, you can specify a different namespace for the catalog to be scoped and available only for that namespace.
- 3 Specify the value of **legacy** or **restricted**. If the field is not set, the default value is **legacy**. In a future OpenShift Container Platform release, it is planned that the default value will be **restricted**.



NOTE

If your catalog cannot run with **restricted** permissions, it is recommended that you manually set this field to **legacy**.

- 4 Specify your index image. If you specify a tag after the image name, for example **:v4.20**, the catalog source pod uses an image pull policy of **Always**, meaning the pod always pulls the image prior to starting the container. If you specify a digest, for example **@sha256:<id>**, the image pull policy is **IfNotPresent**, meaning the pod pulls the image only if it does not already exist on the node.
- 5 Specify your name or an organization name publishing the catalog.
- 6 Catalog sources can automatically check for new versions to keep up to date.

- b. Use the file to create the **CatalogSource** object:

```
$ oc apply -f catalogSource.yaml
```

2. Verify the following resources are created successfully.

- a. Check the pods:

```
$ oc get pods -n openshift-marketplace
```

Example output

```

NAME                                READY STATUS RESTARTS AGE
my-operator-catalog-6njsx6          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 **Software Catalog** page on your OpenShift Container Platform web console.

Additional resources

- [Accessing images for Operators from private registries](#)
- [Image template for custom catalog sources](#)
- [Image pull policy](#)

9.5. ABOUT OPERATOR INSTALLATION FROM THE SOFTWARE CATALOG

The software catalog 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 the software catalog by using the OpenShift Container Platform web console or CLI. Subscribing an Operator to one or more namespaces makes the Operator available to developers on your cluster.

During installation, you must determine the following initial settings for the Operator:

Installation Mode

Choose **All namespaces on the cluster (default)** to have the Operator installed on all namespaces or choose individual namespaces, if available, to only install the Operator on selected namespaces. This example chooses **All namespaces...** to make the Operator available to all users and projects.

Update Channel

If an Operator is available through multiple channels, you can choose which channel you want to subscribe to. For example, to deploy from the **stable** channel, if available, select it from the list.

Approval Strategy

You can choose automatic or manual updates.

If you choose automatic updates for an installed Operator, when a new version of that Operator is available in the selected channel, Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without human intervention.

If you select manual updates, when a newer version of an Operator is available, OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the Operator updated to the new version.

9.5.1. Installing from the software catalog by using the web console

You can install and subscribe to an Operator from software catalog by using the OpenShift Container Platform web console.

Prerequisites

- Access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.

Procedure

1. Navigate in the web console to the **Ecosystem → Software Catalog** 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, configure your Operator installation:
 - a. If you want to install a specific version of an Operator, select an **Update channel** and **Version** from the lists. You can browse the various versions of an Operator across any channels it might have, view the metadata for that channel and version, and select the exact version you want to install.



NOTE

The version selection defaults to the latest version for the channel selected. If the latest version for the channel is selected, the **Automatic** approval strategy is enabled by default. Otherwise, **Manual** approval is required when not installing the latest version for the selected channel.

Installing an Operator with **Manual** approval causes all Operators installed within the namespace to function with the **Manual** approval strategy and all Operators are updated together. If you want to update Operators independently, install Operators into separate namespaces.

- b. Confirm the installation mode for the Operator:

- **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.
- c. For clusters on cloud providers with token authentication enabled:
- If the cluster uses AWS Security Token Service (**STS Mode** in the web console), enter the Amazon Resource Name (ARN) of the AWS IAM role of your service account in the **role ARN** field. To create the role's ARN, follow the procedure described in [Preparing AWS account](#).
 - If the cluster uses Microsoft Entra Workload ID (**Workload Identity / Federated Identity Mode** in the web console), add the client ID, tenant ID, and subscription ID in the appropriate fields.
 - If the cluster uses Google Cloud Platform Workload Identity (**GCP Workload Identity / Federated Identity Mode** in the web console), add the project number, pool ID, provider ID, and service account email in the appropriate fields.
- d. For **Update approval**, select either the **Automatic** or **Manual** approval strategy.



IMPORTANT

If the web console shows that the cluster uses AWS STS, Microsoft Entra Workload ID, or GCP Workload Identity, you must set **Update approval** to **Manual**.

Subscriptions with automatic approvals for updates are not recommended because there might be permission changes to make before updating. Subscriptions with manual approvals for updates ensure that administrators have the opportunity to verify the permissions of the later version, take any necessary steps, and then update.

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.

Verification

- After the upgrade status of the subscription is **Up to date**, select **Ecosystem → Installed Operators** to verify that the cluster service version (CSV) of the installed Operator eventually shows up. The **Status** should eventually resolve to **Succeeded** in the relevant namespace.

**NOTE**

For the **All namespaces...** installation mode, the status resolves to **Succeeded** in the **openshift-operators** namespace, but the status is **Copied** if you check in other namespaces.

If it does not:

- 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.
- When the Operator is installed, the metadata indicates which channel and version are installed.

**NOTE**

The **Channel** and **Version** dropdown menus are still available for viewing other version metadata in this catalog context.

9.5.2. Installing from the software catalog by using the CLI

Instead of using the OpenShift Container Platform web console, you can install an Operator from the software catalog by using the CLI. Use the **oc** command to create or update a **Subscription** object.

For **SingleNamespace** install mode, you must also ensure an appropriate Operator group exists in the related namespace. 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.

TIP

In most cases, the web console method of this procedure is preferred because it automates tasks in the background, such as handling the creation of **OperatorGroup** and **Subscription** objects automatically when choosing **SingleNamespace** mode.

Prerequisites

- Access to your OpenShift Container Platform cluster using an account with **cluster-admin** permissions.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. View the list of Operators available to the cluster from the software catalog:

```
$ oc get packagemanifests -n openshift-marketplace
```

Example 9.1. 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
```

Example 9.2. Example output

```

# ...
Kind:      PackageManifest
# ...
  Install Modes: 1
    Supported: true
    Type:      OwnNamespace
    Supported: true
    Type:      SingleNamespace
    Supported: false
    Type:      MultiNamespace
    Supported: true
    Type:      AllNamespaces
# ...
  Entries:
    Name:      example-operator.v3.7.11
    Version:    3.7.11
    Name:      example-operator.v3.7.10
    Version:    3.7.10
    Name:      stable-3.7 2
# ...
  Entries:
    Name:      example-operator.v3.8.5
    Version:    3.8.5
    Name:      example-operator.v3.8.4
    Version:    3.8.4
    Name:      stable-3.8 3
Default Channel: stable-3.8 4

```

1 Indicates which install modes are supported.

2 **3** Example channel names.

4 The channel selected by default if one is not specified.

TIP

You can print an Operator's version and channel information in YAML format by running the following command:

```
$ oc get packagemanifests <operator_name> -n <catalog_namespace> -o yaml
```

3. If more than one catalog is installed in a namespace, run the following command to look up the available versions and channels of an Operator from a specific catalog:

```
$ oc get packagemanifest \
  --selector=catalog=<catalogsource_name> \
  --field-selector metadata.name=<operator_name> \
  -n <catalog_namespace> -o yaml
```

IMPORTANT

If you do not specify the Operator's catalog, running the **oc get packagemanifest** and **oc describe packagemanifest** commands might return a package from an unexpected catalog if the following conditions are met:

- Multiple catalogs are installed in the same namespace.
- The catalogs contain the same Operators or Operators with the same name.

4. If the Operator you intend to install supports the **AllNamespaces** install mode, and you choose to use this mode, skip this step, because the **openshift-operators** namespace already has an appropriate Operator group in place by default, called **global-operators**.

If the Operator you intend to install supports the **SingleNamespace** install mode, and you choose to use this mode, you must ensure an appropriate Operator group exists in the related namespace. If one does not exist, you can create one by following these steps:

IMPORTANT

You can only have one Operator group per namespace. For more information, see "Operator groups".

- a. Create an **OperatorGroup** object YAML file, for example **operatorgroup.yaml**, for **SingleNamespace** install mode:

Example OperatorGroup object for SingleNamespace install mode

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: <operatorgroup_name>
  namespace: <namespace> 1
spec:
  targetNamespaces:
    - <namespace> 2
```

- 1 2** For **SingleNamespace** install mode, use the same **<namespace>** value for both the **metadata.namespace** and **spec.targetNamespaces** fields.

- b. Create the **OperatorGroup** object:

```
$ oc apply -f operatorgroup.yaml
```

5. Create a **Subscription** object to subscribe a namespace to an Operator:

- a. Create a YAML file for the **Subscription** object, for example **subscription.yaml**:



NOTE

If you want to subscribe to a specific version of an Operator, set the **startingCSV** field to the desired version and set the **installPlanApproval** field to **Manual** to prevent the Operator from automatically upgrading if a later version exists in the catalog. For details, see the following "Example **Subscription** object with a specific starting Operator version".

Example 9.3. Example **Subscription** object

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: <subscription_name>
  namespace: <namespace_per_install_mode> 1
spec:
  channel: <channel_name> 2
  name: <operator_name> 3
  source: <catalog_name> 4
  sourceNamespace: <catalog_source_namespace> 5
  config:
    env: 6
    - name: ARGS
      value: "-v=10"
    envFrom: 7
    - secretRef:
        name: license-secret
  volumes: 8
  - name: <volume_name>
    configMap:
      name: <configmap_name>
  volumeMounts: 9
  - mountPath: <directory_name>
    name: <volume_name>
  tolerations: 10
  - operator: "Exists"
  resources: 11
    requests:
      memory: "64Mi"
      cpu: "250m"
    limits:
```

```
memory: "128Mi"
cpu: "500m"
nodeSelector: 12
foo: bar
```

- 1 For default **AllNamespaces** install mode usage, specify the **openshift-operators** namespace. Alternatively, you can specify a custom global namespace, if you have created one. For **SingleNamespace** install mode usage, specify the relevant single namespace.
- 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 software catalog sources.
- 6 The **env** parameter defines a list of environment variables that must exist in all containers in the pod created by OLM.
- 7 The **envFrom** parameter defines a list of sources to populate environment variables in the container.
- 8 The **volumes** parameter defines a list of volumes that must exist on the pod created by OLM.
- 9 The **volumeMounts** parameter defines a list of volume mounts that must exist in all containers in the pod created by OLM. If a **volumeMount** references a **volume** that does not exist, OLM fails to deploy the Operator.
- 10 The **tolerations** parameter defines a list of tolerations for the pod created by OLM.
- 11 The **resources** parameter defines resource constraints for all the containers in the pod created by OLM.
- 12 The **nodeSelector** parameter defines a **NodeSelector** for the pod created by OLM.

Example 9.4. Example **Subscription** object with a specific starting Operator version

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: example-operator
  namespace: example-operator
spec:
  channel: stable-3.7
  installPlanApproval: Manual 1
  name: example-operator
```

```
source: custom-operators
sourceNamespace: openshift-marketplace
startingCSV: example-operator.v3.7.10 2
```

- 1** Set the approval strategy to **Manual** in case your specified version is superseded by a later version in the catalog. This plan prevents an automatic upgrade to a later version and requires manual approval before the starting CSV can complete the installation.
 - 2** Set a specific version of an Operator CSV.
- b. For clusters on cloud providers with token authentication enabled, such as Amazon Web Services (AWS) Security Token Service (STS), Microsoft Entra Workload ID, or Google Cloud Platform Workload Identity, configure your **Subscription** object by following these steps:

- i. Ensure the **Subscription** object is set to manual update approvals:

Example 9.5. Example **Subscription** object with manual update approvals

```
kind: Subscription
# ...
spec:
  installPlanApproval: Manual 1
```

- 1** Subscriptions with automatic approvals for updates are not recommended because there might be permission changes to make before updating. Subscriptions with manual approvals for updates ensure that administrators have the opportunity to verify the permissions of the later version, take any necessary steps, and then update.

- ii. Include the relevant cloud provider-specific fields in the **Subscription** object's **config** section:

If the cluster is in AWS STS mode, include the following fields:

Example 9.6. Example **Subscription** object with AWS STS variables

```
kind: Subscription
# ...
spec:
  config:
    env:
      - name: ROLEARN
        value: "<role_arn>" 1
```

- 1** Include the role ARN details.

If the cluster is in Workload ID mode, include the following fields:

Example 9.7. Example `Subscription` object with Workload ID variables

```

kind: Subscription
# ...
spec:
  config:
    env:
      - name: CLIENTID
        value: "<client_id>" 1
      - name: TENANTID
        value: "<tenant_id>" 2
      - name: SUBSCRIPTIONID
        value: "<subscription_id>" 3

```

- 1 Include the client ID.
- 2 Include the tenant ID.
- 3 Include the subscription ID.

If the cluster is in GCP Workload Identity mode, include the following fields:

Example 9.8. Example `Subscription` object with GCP Workload Identity variables

```

kind: Subscription
# ...
spec:
  config:
    env:
      - name: AUDIENCE
        value: "<audience_url>" 1
      - name: SERVICE_ACCOUNT_EMAIL
        value: "<service_account_email>" 2

```

where:

<audience>

Created in Google Cloud by the administrator when they set up GCP Workload Identity, the **AUDIENCE** value must be a preformatted URL in the following format:

```
//iam.googleapis.com/projects/<project_number>/locations/global/workloadIdentityPools/<pool_id>/providers/<provider_id>
```

<service_account_email>

The **SERVICE_ACCOUNT_EMAIL** value is a Google Cloud service account email that is impersonated during Operator operation, for example:

```
<service_account_name>@<project_id>.iam.gserviceaccount.com
```

- c. Create the **Subscription** object by running the following command:

```
$ oc apply -f subscription.yaml
```

6. If you set the **installPlanApproval** field to **Manual**, manually approve the pending install plan to complete the Operator installation. For more information, see "Manually approving a pending Operator update".

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.

Verification

1. Check the status of the **Subscription** object for your installed Operator by running the following command:

```
$ oc describe subscription <subscription_name> -n <namespace>
```

2. If you created an Operator group for **SingleNamespace** install mode, check the status of the **OperatorGroup** object by running the following command:

```
$ oc describe operatorgroup <operatorgroup_name> -n <namespace>
```

Additional resources

- [About OperatorGroups](#)

CHAPTER 10. CHANGING THE CLOUD PROVIDER CREDENTIALS CONFIGURATION

For supported configurations, you can change how OpenShift Container Platform authenticates with your cloud provider.

To determine which cloud credentials strategy your cluster uses, see [Determining the Cloud Credential Operator mode](#).

10.1. ROTATING CLOUD PROVIDER SERVICE KEYS WITH THE CLOUD CREDENTIAL OPERATOR UTILITY

Some organizations require the rotation of the service keys that authenticate the cluster. You can use the Cloud Credential Operator (CCO) utility (**ccoctl**) to update keys for clusters installed on the following cloud providers:

- [Amazon Web Services \(AWS\) with Security Token Service \(STS\)](#)
- [Google Cloud with GCP Workload Identity](#)
- [Microsoft Azure with Workload ID](#)
- [IBM Cloud](#)

10.1.1. Rotating AWS OIDC bound service account signer keys

If the Cloud Credential Operator (CCO) for your OpenShift Container Platform cluster on Amazon Web Services (AWS) is configured to operate in manual mode with STS, you can rotate the bound service account signer key.

To rotate the key, you delete the existing key on your cluster, which causes the Kubernetes API server to create a new key. To reduce authentication failures during this process, you must immediately add the new public key to the existing issuer file. After the cluster is using the new key for authentication, you can remove any remaining keys.



IMPORTANT

The process to rotate OIDC bound service account signer keys is disruptive and takes a significant amount of time. Some steps are time-sensitive. Before proceeding, observe the following considerations:

- Read the following steps and ensure that you understand and accept the time requirement. The exact time requirement varies depending on the individual cluster, but it is likely to require at least one hour.
- To reduce the risk of authentication failures, ensure that you understand and prepare for the time-sensitive steps.
- During this process, you must refresh all service accounts and restart all pods on the cluster. These actions are disruptive to workloads. To mitigate this impact, you can temporarily halt these services and then redeploy them when the cluster is ready.

Prerequisites

- You have access to the OpenShift CLI (**oc**) as a user with the **cluster-admin** role.
- You have created an AWS account for the **ccoctl** utility to use with the following permissions:
 - **s3:GetObject**
 - **s3:PutObject**
 - **s3:PutObjectTagging**
 - For clusters that store the OIDC configuration in a private S3 bucket that is accessed by the IAM identity provider through a public CloudFront distribution URL, the AWS account that runs the **ccoctl** utility requires the **cloudfront:ListDistributions** permission.
- You have configured the **ccoctl** utility.
- Your cluster is in a stable state. You can confirm that the cluster is stable by running the following command:

```
$ oc adm wait-for-stable-cluster --minimum-stable-period=5s
```

Procedure

1. Configure the following environment variables:

```
INFRA_ID=$(oc get infrastructures cluster -o jsonpath='{.status.infrastructureName}')
CLUSTER_NAME=${INFRA_ID%-*} 1
```

- 1 1 This value should match the name of the cluster that was specified in the **metadata.name** field of the **install-config.yaml** file during installation.



NOTE

Your cluster might differ from this example, and the resource names might not be derived identically from the cluster name. Ensure that you specify the correct corresponding resource names for your cluster.

- For AWS clusters that store the OIDC configuration in a public S3 bucket, configure the following environment variable:

```
AWS_BUCKET=$(oc get authentication cluster -o jsonpath=
{'spec.serviceAccountIssuer'} | awk -F:/' '{print$2}' |awk -F.' '{print$1}')
```

- For AWS clusters that store the OIDC configuration in a private S3 bucket that is accessed by the IAM identity provider through a public CloudFront distribution URL, complete the following steps:
 - i. Extract the public CloudFront distribution URL by running the following command:

```
$ basename $(oc get authentication cluster -o jsonpath=
{'spec.serviceAccountIssuer'})
```

Example output

```
<subdomain>.cloudfront.net
```

where **<subdomain>** is an alphanumeric string.

- ii. Determine the private S3 bucket name by running the following command:

```
$ aws cloudfront list-distributions --query "DistributionList.Items[?{DomainName:
DomainName, OriginDomainName: Origins.Items[0].DomainName}][?
contains(DomainName, '<subdomain>.cloudfront.net')]"
```

Example output

```
[
  {
    "DomainName": "<subdomain>.cloudfront.net",
    "OriginDomainName": "<s3_bucket>.s3.us-east-2.amazonaws.com"
  }
]
```

where **<s3_bucket>** is the private S3 bucket name for your cluster.

- iii. Configure the following environment variable:

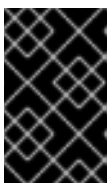
```
AWS_BUCKET=$<s3_bucket>
```

where **<s3_bucket>** is the private S3 bucket name for your cluster.

2. Create a temporary directory to use and assign it an environment variable by running the following command:

```
$ TEMPDIR=$(mktemp -d)
```

3. To cause the Kubernetes API server to create a new bound service account signing key, you delete the next bound service account signing key.



IMPORTANT

After you complete this step, the Kubernetes API server starts to roll out a new key. To reduce the risk of authentication failures, complete the remaining steps as quickly as possible. The remaining steps might be disruptive to workloads.

When you are ready, delete the next bound service account signing key by running the following command:

```
$ oc delete secrets/next-bound-service-account-signing-key \
-n openshift-kube-apiserver-operator
```

4. Download the public key from the service account signing key secret that the Kubernetes API server created by running the following command:

```
$ oc get secret/next-bound-service-account-signing-key \
-n openshift-kube-apiserver-operator \
```

```
-ojsonpath='{ .data.service-account\.pub }' | base64 \
-d > ${TEMPDIR}/serviceaccount-signer.public
```

- Use the public key to create a **keys.json** file by running the following command:

```
$ ccoctl aws create-identity-provider \
  --dry-run 1 \
  --output-dir ${TEMPDIR} \
  --public-key-file=${TEMPDIR}/serviceaccount-signer.public 2 \
  --name fake 3 \
  --region us-east-1 4
```

- The **--dry-run** option outputs files, including the new **keys.json** file, to the disk without making API calls.
- Specify the path to the public key that you downloaded in the previous step.
- Because the **--dry-run** option does not make any API calls, some parameters do not require real values.
- Specify any valid AWS region, such as **us-east-1**. This value does not need to match the region the cluster is in.

- Rename the **keys.json** file by running the following command:

```
$ cp ${TEMPDIR}/<number>-keys.json ${TEMPDIR}/jwks.new.json
```

where **<number>** is a two-digit numerical value that varies depending on your environment.

- Download the existing **keys.json** file from the cloud provider by running the following command:

```
$ aws s3api get-object \
  --bucket ${AWS_BUCKET} \
  --key keys.json ${TEMPDIR}/jwks.current.json
```

- Combine the two **keys.json** files by running the following command:

```
$ jq -s '{ keys: map(.keys[])}' ${TEMPDIR}/jwks.current.json ${TEMPDIR}/jwks.new.json >
${TEMPDIR}/jwks.combined.json
```

- To enable authentication for the old and new keys during the rotation, upload the combined **keys.json** file to the cloud provider by running the following command:

```
$ aws s3api put-object \
  --bucket ${AWS_BUCKET} \
  --tagging "openshift.io/cloud-credential-operator/${CLUSTER_NAME}=owned" \
  --key keys.json \
  --body ${TEMPDIR}/jwks.combined.json
```

- Wait for the Kubernetes API server to update and use the new key. You can monitor the update progress by running the following command:

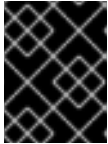
—

```
$ oc adm wait-for-stable-cluster
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All clusteroperators are stable
```

11. To ensure that all pods on the cluster use the new key, you must restart them.



IMPORTANT

This step maintains uptime for services that are configured for high availability across multiple nodes, but might cause downtime for any services that are not.

Restart all of the pods in the cluster by running the following command:

```
$ oc adm reboot-machine-config-pool mcp/worker mcp/master
```

12. Monitor the restart and update process by running the following command:

```
$ oc adm wait-for-node-reboot nodes --all
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All nodes rebooted
```

13. Monitor the update progress by running the following command:

```
$ oc adm wait-for-stable-cluster
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All clusteroperators are stable
```

14. Replace the combined **keys.json** file with the updated **keys.json** file on the cloud provider by running the following command:

```
$ aws s3api put-object \
  --bucket ${AWS_BUCKET} \
  --tagging "openshift.io/cloud-credential-operator/${CLUSTER_NAME}=owned" \
  --key keys.json \
  --body ${TEMPDIR}/jwks.new.json
```

10.1.2. Rotating Google Cloud OIDC bound service account signer keys

If the Cloud Credential Operator (CCO) for your OpenShift Container Platform cluster on Google Cloud is configured to operate in manual mode with GCP Workload Identity, you can rotate the bound service account signer key.

To rotate the key, you delete the existing key on your cluster, which causes the Kubernetes API server to create a new key. To reduce authentication failures during this process, you must immediately add the new public key to the existing issuer file. After the cluster is using the new key for authentication, you can remove any remaining keys.



IMPORTANT

The process to rotate OIDC bound service account signer keys is disruptive and takes a significant amount of time. Some steps are time-sensitive. Before proceeding, observe the following considerations:

- Read the following steps and ensure that you understand and accept the time requirement. The exact time requirement varies depending on the individual cluster, but it is likely to require at least one hour.
- To reduce the risk of authentication failures, ensure that you understand and prepare for the time-sensitive steps.
- During this process, you must refresh all service accounts and restart all pods on the cluster. These actions are disruptive to workloads. To mitigate this impact, you can temporarily halt these services and then redeploy them when the cluster is ready.

Prerequisites

- You have access to the OpenShift CLI (**oc**) as a user with the **cluster-admin** role.
- You have added one of the following authentication options to the Google Cloud account that the **ccoctl** utility uses:
 - The **IAM Workload Identity Pool Admin** role
 - The following granular permissions:
 - **storage.objects.create**
 - **storage.objects.delete**
- You have configured the **ccoctl** utility.
- Your cluster is in a stable state. You can confirm that the cluster is stable by running the following command:

```
$ oc adm wait-for-stable-cluster --minimum-stable-period=5s
```

Procedure

1. Configure the following environment variables:

```
CURRENT_ISSUER=$(oc get authentication cluster -o
jsonpath='{.spec.serviceAccountIssuer}')
GCP_BUCKET=$(echo ${CURRENT_ISSUER} | cut -d "/" -f4)
```

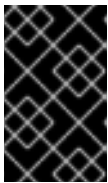
**NOTE**

Your cluster might differ from this example, and the resource names might not be derived identically from the cluster name. Ensure that you specify the correct corresponding resource names for your cluster.

2. Create a temporary directory to use and assign it an environment variable by running the following command:

```
$ TEMPDIR=$(mktemp -d)
```

3. To cause the Kubernetes API server to create a new bound service account signing key, you delete the next bound service account signing key.

**IMPORTANT**

After you complete this step, the Kubernetes API server starts to roll out a new key. To reduce the risk of authentication failures, complete the remaining steps as quickly as possible. The remaining steps might be disruptive to workloads.

When you are ready, delete the next bound service account signing key by running the following command:

```
$ oc delete secrets/next-bound-service-account-signing-key \
  -n openshift-kube-apiserver-operator
```

4. Download the public key from the service account signing key secret that the Kubernetes API server created by running the following command:

```
$ oc get secret/next-bound-service-account-signing-key \
  -n openshift-kube-apiserver-operator \
  -ojsonpath='{.data.service-account\.pub}' | base64 \
  -d > ${TEMPDIR}/serviceaccount-signer.public
```

5. Use the public key to create a **keys.json** file by running the following command:

```
$ ccoctl gcp create-workload-identity-provider \
  --dry-run \ ❶
  --output-dir=${TEMPDIR} \
  --public-key-file=${TEMPDIR}/serviceaccount-signer.public \ ❷
  --name fake \ ❸
  --project fake \
  --workload-identity-pool fake
```

- ❶ The **--dry-run** option outputs files, including the new **keys.json** file, to the disk without making API calls.
- ❷ Specify the path to the public key that you downloaded in the previous step.
- ❸ Because the **--dry-run** option does not make any API calls, some parameters do not require real values.

- Rename the **keys.json** file by running the following command:

```
$ cp ${TEMPDIR}/<number>-keys.json ${TEMPDIR}/jwks.new.json
```

where **<number>** is a two-digit numerical value that varies depending on your environment.

- Download the existing **keys.json** file from the cloud provider by running the following command:

```
$ gcloud storage cp gs://${GCP_BUCKET}/keys.json ${TEMPDIR}/jwks.current.json
```

- Combine the two **keys.json** files by running the following command:

```
$ jq -s '{ keys: map(.keys[])}' ${TEMPDIR}/jwks.current.json ${TEMPDIR}/jwks.new.json >
${TEMPDIR}/jwks.combined.json
```

- To enable authentication for the old and new keys during the rotation, upload the combined **keys.json** file to the cloud provider by running the following command:

```
$ gcloud storage cp ${TEMPDIR}/jwks.combined.json gs://${GCP_BUCKET}/keys.json
```

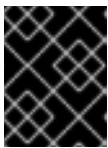
- Wait for the Kubernetes API server to update and use the new key. You can monitor the update progress by running the following command:

```
$ oc adm wait-for-stable-cluster
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All clusteroperators are stable
```

- To ensure that all pods on the cluster use the new key, you must restart them.



IMPORTANT

This step maintains uptime for services that are configured for high availability across multiple nodes, but might cause downtime for any services that are not.

Restart all of the pods in the cluster by running the following command:

```
$ oc adm reboot-machine-config-pool mcp/worker mcp/master
```

- Monitor the restart and update process by running the following command:

```
$ oc adm wait-for-node-reboot nodes --all
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All nodes rebooted
```


13. Monitor the update progress by running the following command:

```
$ oc adm wait-for-stable-cluster
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All clusteroperators are stable
```

14. Replace the combined **keys.json** file with the updated **keys.json** file on the cloud provider by running the following command:

```
$ gcloud storage cp ${TEMPDIR}/jwks.new.json gs://${GCP_BUCKET}/keys.json
```

10.1.3. Rotating Azure OIDC bound service account signer keys

If the Cloud Credential Operator (CCO) for your OpenShift Container Platform cluster on Microsoft Azure is configured to operate in manual mode with Microsoft Entra Workload ID, you can rotate the bound service account signer key.

To rotate the key, you delete the existing key on your cluster, which causes the Kubernetes API server to create a new key. To reduce authentication failures during this process, you must immediately add the new public key to the existing issuer file. After the cluster is using the new key for authentication, you can remove any remaining keys.

IMPORTANT

The process to rotate OIDC bound service account signer keys is disruptive and takes a significant amount of time. Some steps are time-sensitive. Before proceeding, observe the following considerations:

- Read the following steps and ensure that you understand and accept the time requirement. The exact time requirement varies depending on the individual cluster, but it is likely to require at least one hour.
- To reduce the risk of authentication failures, ensure that you understand and prepare for the time-sensitive steps.
- During this process, you must refresh all service accounts and restart all pods on the cluster. These actions are disruptive to workloads. To mitigate this impact, you can temporarily halt these services and then redeploy them when the cluster is ready.

Prerequisites

- You have access to the OpenShift CLI (**oc**) as a user with the **cluster-admin** role.
- You have created a global Azure account for the **ccoctl** utility to use with the following permissions:
 - **Microsoft.Storage/storageAccounts/listkeys/action**
 - **Microsoft.Storage/storageAccounts/read**

- **Microsoft.Storage/storageAccounts/write**
- **Microsoft.Storage/storageAccounts/blobServices/containers/read**
- **Microsoft.Storage/storageAccounts/blobServices/containers/write**
- You have configured the **ccctl** utility.
- Your cluster is in a stable state. You can confirm that the cluster is stable by running the following command:

```
$ oc adm wait-for-stable-cluster --minimum-stable-period=5s
```

Procedure

1. Configure the following environment variables:

```
CURRENT_ISSUER=$(oc get authentication cluster -o
jsonpath='{.spec.serviceAccountIssuer}')
AZURE_STORAGE_ACCOUNT=$(echo ${CURRENT_ISSUER} | cut -d "/" -f3 | cut -d "." -f1)
AZURE_STORAGE_CONTAINER=$(echo ${CURRENT_ISSUER} | cut -d "/" -f4)
```



NOTE

Your cluster might differ from this example, and the resource names might not be derived identically from the cluster name. Ensure that you specify the correct corresponding resource names for your cluster.

2. Create a temporary directory to use and assign it an environment variable by running the following command:

```
$ TEMPDIR=$(mktemp -d)
```

3. To cause the Kubernetes API server to create a new bound service account signing key, you delete the next bound service account signing key.



IMPORTANT

After you complete this step, the Kubernetes API server starts to roll out a new key. To reduce the risk of authentication failures, complete the remaining steps as quickly as possible. The remaining steps might be disruptive to workloads.

When you are ready, delete the next bound service account signing key by running the following command:

```
$ oc delete secrets/next-bound-service-account-signing-key \
-n openshift-kube-apiserver-operator
```

4. Download the public key from the service account signing key secret that the Kubernetes API server created by running the following command:

```
$ oc get secret/next-bound-service-account-signing-key \
```

```
-n openshift-kube-apiserver-operator \
-ojsonpath='{ .data.service-account\.pub }' | base64 \
-d > ${TEMPDIR}/serviceaccount-signer.public
```

5. Use the public key to create a **keys.json** file by running the following command:

```
$ ccoctl aws create-identity-provider \ ❶
--dry-run \ ❷
--output-dir ${TEMPDIR} \
--public-key-file=${TEMPDIR}/serviceaccount-signer.public \ ❸
--name fake \ ❹
--region us-east-1 ❺
```

- ❶ The **ccoctl azure** command does not include a **--dry-run** option. To use the **--dry-run** option, you must specify **aws** for an Azure cluster.
- ❷ The **--dry-run** option outputs files, including the new **keys.json** file, to the disk without making API calls.
- ❸ Specify the path to the public key that you downloaded in the previous step.
- ❹ Because the **--dry-run** option does not make any API calls, some parameters do not require real values.
- ❺ Specify any valid AWS region, such as **us-east-1**. This value does not need to match the region the cluster is in.

6. Rename the **keys.json** file by running the following command:

```
$ cp ${TEMPDIR}/<number>-keys.json ${TEMPDIR}/jwks.new.json
```

where **<number>** is a two-digit numerical value that varies depending on your environment.

7. Download the existing **keys.json** file from the cloud provider by running the following command:

```
$ az storage blob download \
--container-name ${AZURE_STORAGE_CONTAINER} \
--account-name ${AZURE_STORAGE_ACCOUNT} \
--name 'openid/v1/jwks' \
-f ${TEMPDIR}/jwks.current.json
```

8. Combine the two **keys.json** files by running the following command:

```
$ jq -s '{ keys: map(.keys[])}' ${TEMPDIR}/jwks.current.json ${TEMPDIR}/jwks.new.json >
${TEMPDIR}/jwks.combined.json
```

9. To enable authentication for the old and new keys during the rotation, upload the combined **keys.json** file to the cloud provider by running the following command:

```
$ az storage blob upload \
--overwrite \
--account-name ${AZURE_STORAGE_ACCOUNT} \
```

```
--container-name ${AZURE_STORAGE_CONTAINER} \
--name 'openid/v1/jwks' \
-f ${TEMPDIR}/jwks.combined.json
```

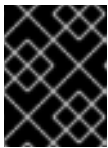
10. Wait for the Kubernetes API server to update and use the new key. You can monitor the update progress by running the following command:

```
$ oc adm wait-for-stable-cluster
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All clusteroperators are stable
```

11. To ensure that all pods on the cluster use the new key, you must restart them.



IMPORTANT

This step maintains uptime for services that are configured for high availability across multiple nodes, but might cause downtime for any services that are not.

Restart all of the pods in the cluster by running the following command:

```
$ oc adm reboot-machine-config-pool mcp/worker mcp/master
```

12. Monitor the restart and update process by running the following command:

```
$ oc adm wait-for-node-reboot nodes --all
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All nodes rebooted
```

13. Monitor the update progress by running the following command:

```
$ oc adm wait-for-stable-cluster
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All clusteroperators are stable
```

14. Replace the combined **keys.json** file with the updated **keys.json** file on the cloud provider by running the following command:

```
$ az storage blob upload \
--overwrite \
--account-name ${AZURE_STORAGE_ACCOUNT} \
--container-name ${AZURE_STORAGE_CONTAINER} \
--name 'openid/v1/jwks' \
-f ${TEMPDIR}/jwks.new.json
```

10.1.4. Rotating IBM Cloud credentials

You can rotate API keys for your existing service IDs and update the corresponding secrets.

Prerequisites

- You have configured the **ccctl** utility.
- You have existing service IDs in a live OpenShift Container Platform cluster installed.

Procedure

- Use the **ccctl** utility to rotate your API keys for the service IDs and update the secrets by running the following command:

```
$ ccctl <provider_name> refresh-keys \1
--kubeconfig <openshift_kubeconfig_file> \2
--credentials-requests-dir <path_to_credential_requests_directory> \3
--name <name> \4
```

- 1 The name of the provider. For example: **ibmcloud** or **powervs**.
- 2 The **kubeconfig** file associated with the cluster. For example, **<installation_directory>/auth/kubeconfig**.
- 3 The directory where the credential requests are stored.
- 4 The name of the OpenShift Container Platform cluster.



NOTE

If your cluster uses Technology Preview features that are enabled by the **TechPreviewNoUpgrade** feature set, you must include the **--enable-tech-preview** parameter.

10.2. ROTATING CLOUD PROVIDER CREDENTIALS

Some organizations require the rotation of the cloud provider credentials. 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.

10.2.1. 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, Amazon Web Services (AWS) and Google Cloud are supported.
 - For passthrough mode, Amazon Web Services (AWS), Microsoft Azure, Google Cloud, Red Hat OpenStack Platform (RHOSP), and VMware vSphere are supported.
- You have changed the credentials that are used to interface with your cloud provider.
- The new credentials have sufficient permissions for the mode CCO is configured to use in your cluster.

Procedure

1. In the **Administrator** perspective of the web console, navigate to **Workloads → Secrets**.
2. In the table on the **Secrets** page, find the root secret for your cloud provider.

Platform	Secret name
AWS	aws-creds
Azure	azure-credentials
Google Cloud	gcp-credentials
RHOSP	openstack-credentials
VMware vSphere	vsphere-creds

3. Click the Options menu  in the same row as the secret and select **Edit Secret**.
4. Record the contents of the **Value** field or fields. You can use this information to verify that the value is different after updating the credentials.
5. Update the text in the **Value** field or fields with the new authentication information for your cloud provider, and then click **Save**.
6. If you are updating the credentials for a vSphere cluster that does not have the vSphere CSI Driver Operator enabled, you must force a rollout of the Kubernetes controller manager to apply the updated credentials.



NOTE

If the vSphere CSI Driver Operator is enabled, this step is not required.

To apply the updated vSphere credentials, log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role and run the following command:

```
$ oc patch kubecontrollermanager cluster \
  -p='{ "spec": { "forceRedeploymentReason": "recovery-"'$( date )'"' } }' \
  --type=merge
```

While the credentials are rolling out, the status of the Kubernetes Controller Manager Operator reports **Progressing=true**. To view the status, run the following command:

```
$ oc get co kube-controller-manager
```

7. If the CCO for your cluster is configured to use mint mode, delete each component secret that is referenced by the individual **CredentialsRequest** objects.

- a. Log in to the OpenShift Container Platform CLI as a user with the **cluster-admin** role.

- b. Get the names and namespaces of all referenced component secrets:

```
$ oc -n openshift-cloud-credential-operator get CredentialsRequest \
  -o json | jq -r '.items[] | select (.spec.providerSpec.kind=="<provider_spec>") | \
  .spec.secretRef'
```

where **<provider_spec>** is the corresponding value for your cloud provider:

- AWS: **AWSProviderSpec**
- Google Cloud: **GCPPProviderSpec**

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> 1 \
  -n <secret_namespace> 2
```

1 Specify the name of a secret.

2 Specify the namespace that contains the secret.

Example deletion of an AWS secret

```
$ oc delete secret ebs-cloud-credentials -n openshift-cluster-csi-drivers
```

You do not need to manually delete the credentials from your provider console. Deleting the referenced component secrets will cause the CCO to delete the existing credentials from the platform and create new ones.

Verification

To verify that the credentials have changed:

1. In the **Administrator** perspective of the web console, navigate to **Workloads → Secrets**.
2. Verify that the contents of the **Value** field or fields have changed.

Additional resources

- [The Cloud Credential Operator in mint mode](#)
- [The Cloud Credential Operator in passthrough mode](#)
- [vSphere CSI Driver Operator](#)

10.3. REMOVING CLOUD PROVIDER CREDENTIALS

After installing OpenShift Container Platform, some organizations require the 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.

10.3.1. Removing cloud provider credentials

For clusters that use the Cloud Credential Operator (CCO) in mint mode, the administrator-level credential is stored in the **kube-system** namespace. The CCO uses the **admin** credential to process the **CredentialsRequest** objects in the cluster and create users for components with limited permissions.

After installing an OpenShift Container Platform cluster with the CCO in mint mode, you can remove the administrator-level credential secret from the **kube-system** namespace in the cluster. The CCO only requires the administrator-level credential during changes that require reconciling new or modified **CredentialsRequest** custom resources, such as minor cluster version updates.



NOTE

Before performing a minor version cluster update (for example, updating from OpenShift Container Platform 4.19 to 4.20), you must reinstate the credential secret with the administrator-level credential. If the credential is not present, the update might be blocked.

Prerequisites

- Your cluster is installed on a platform that supports removing cloud credentials from the CCO. Supported platforms are AWS and Google Cloud.

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
Google Cloud	gcp-credentials

- Click the Options menu  in the same row as the secret and select **Delete Secret**

Additional resources

- [The Cloud Credential Operator in mint mode](#)

10.4. ENABLING TOKEN-BASED AUTHENTICATION

After installing an OpenShift Container Platform cluster on Microsoft Azure or Amazon Web Services (AWS), you can enable Microsoft Entra Workload ID or Security Token Service (STS) to use short-term credentials.

10.4.1. Configuring the Cloud Credential Operator utility

To configure an existing cluster to create and manage cloud credentials from outside of the cluster, extract and prepare the Cloud Credential Operator utility (**ccctl**) binary.



NOTE

The **ccctl** utility is a Linux binary that must run in a Linux environment.

Prerequisites

- You have access to an OpenShift Container Platform account with cluster administrator access.
- You have installed the OpenShift CLI (**oc**).

Procedure

- Set a variable for the OpenShift Container Platform release image by running the following command:

```
$ RELEASE_IMAGE=$(oc get clusterversion -o jsonpath={..desired.image})
```

- Obtain the CCO container image from the OpenShift Container Platform release image by running the following command:

```
$ CCO_IMAGE=$(oc adm release info --image-for='cloud-credential-operator'
$RELEASE_IMAGE -a ~/.pull-secret)
```

**NOTE**

Ensure that the architecture of the **\$RELEASE_IMAGE** matches the architecture of the environment in which you will use the **ccoctl** tool.

3. Extract the **ccoctl** binary from the CCO container image within the OpenShift Container Platform release image by running the following command:

```
$ oc image extract $CCO_IMAGE \
  --file="/usr/bin/ccoctl.<rhel_version>" \
  -a ~/.pull-secret
```

- 1 For **<rhel_version>**, specify the value that corresponds to the version of Red Hat Enterprise Linux (RHEL) that the host uses. If no value is specified, **ccoctl.rhel8** is used by default. The following values are valid:

- **rhel8**: Specify this value for hosts that use RHEL 8.
- **rhel9**: Specify this value for hosts that use RHEL 9.

**NOTE**

The **ccoctl** binary is created in the directory from where you executed the command and not in **/usr/bin/**. You must rename the directory or move the **ccoctl.<rhel_version>** binary to **ccoctl**.

4. Change the permissions to make **ccoctl** executable by running the following command:

```
$ chmod 775 ccoctl
```

Verification

- To verify that **ccoctl** is ready to use, display the help file. Use a relative file name when you run the command, for example:

```
$ ./ccoctl
```

Example output

```
OpenShift credentials provisioning tool

Usage:
  ccoctl [command]

Available Commands:
  aws      Manage credentials objects for AWS cloud
  azure    Manage credentials objects for Azure
  gcp      Manage credentials objects for Google cloud
  help     Help about any command
  ibmcloud Manage credentials objects for {ibm-cloud-title}
  nutanix  Manage credentials objects for Nutanix
```

Flags:`-h, --help` help for `ccoctl`Use "`ccoctl [command] --help`" for more information about a command.

10.4.2. Enabling Microsoft Entra Workload ID on an existing cluster

If you did not configure your Microsoft Azure OpenShift Container Platform cluster to use Microsoft Entra Workload ID during installation, you can enable this authentication method on an existing cluster.

IMPORTANT

The process to enable Workload ID on an existing cluster is disruptive and takes a significant amount of time. Before proceeding, observe the following considerations:

- Read the following steps and ensure that you understand and accept the time requirement. The exact time requirement varies depending on the individual cluster, but it is likely to require at least one hour.
- During this process, you must refresh all service accounts and restart all pods on the cluster. These actions are disruptive to workloads. To mitigate this impact, you can temporarily halt these services and then redeploy them when the cluster is ready.
- After starting this process, do not attempt to update the cluster until it is complete. If an update is triggered, the process to enable Workload ID on an existing cluster fails.

Prerequisites

- You have installed an OpenShift Container Platform cluster on Microsoft Azure.
- You have access to the cluster using an account with **cluster-admin** permissions.
- You have installed the OpenShift CLI (**oc**).
- You have extracted and prepared the Cloud Credential Operator utility (**ccoctl**) binary.
- You have access to your Azure account by using the Azure CLI (**az**).

Procedure

1. Create an output directory for the manifests that the **ccoctl** utility generates. This procedure uses **./output_dir** as an example.
2. Extract the service account public signing key for the cluster to the output directory by running the following command:

```
$ oc get secret/next-bound-service-account-signing-key \
  -n openshift-kube-apiserver-operator \
  -ojsonpath='{ .data.service-account\.pub }' | base64 -d \
  > output_dir/serviceaccount-signer.public 1
```

1 This procedure uses a file named **serviceaccount-signer.public** as an example.

- Use the extracted service account public signing key to create an OpenID Connect (OIDC) issuer and Azure blob storage container with OIDC configuration files by running the following command:

```
$ ./ccoctl azure create-oidc-issuer \
  --name <azure_infra_name> \ 1
  --output-dir ./output_dir \
  --region <azure_region> \ 2
  --subscription-id <azure_subscription_id> \ 3
  --tenant-id <azure_tenant_id> \
  --public-key-file ./output_dir/serviceaccount-signer.public 4
```

- The value of the **name** parameter is used to create an Azure resource group. To use an existing Azure resource group instead of creating a new one, specify the **--oidc-resource-group-name** argument with the existing group name as its value.
- Specify the region of the existing cluster.
- Specify the subscription ID of the existing cluster.
- Specify the file that contains the service account public signing key for the cluster.

- Verify that the configuration file for the Azure pod identity webhook was created by running the following command:

```
$ ll ./output_dir/manifests
```

Example output

```
total 8
-rw-----. 1 cloud-user cloud-user 193 May 22 02:29 azure-ad-pod-identity-webhook-
config.yaml 1
-rw-----. 1 cloud-user cloud-user 165 May 22 02:29 cluster-authentication-02-config.yaml
```

- The file **azure-ad-pod-identity-webhook-config.yaml** contains the Azure pod identity webhook configuration.

- Set an **OIDC_ISSUER_URL** variable with the OIDC issuer URL from the generated manifests in the output directory by running the following command:

```
$ OIDC_ISSUER_URL=`awk '/serviceAccountIssuer/ { print $2 }'
./output_dir/manifests/cluster-authentication-02-config.yaml`
```

- Update the **spec.serviceAccountIssuer** parameter of the cluster **authentication** configuration by running the following command:

```
$ oc patch authentication cluster \
  --type=merge \
  -p '{"spec":{"serviceAccountIssuer":"'${OIDC_ISSUER_URL}'"}}'
```

- Monitor the configuration update progress by running the following command:

```
$ oc adm wait-for-stable-cluster
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All clusteroperators are stable
```

8. Restart all of the pods in the cluster by running the following command:

```
$ oc adm reboot-machine-config-pool mcp/worker mcp/master
```

Restarting a pod updates the **serviceAccountIssuer** field and refreshes the service account public signing key.

9. Monitor the restart and update process by running the following command:

```
$ oc adm wait-for-node-reboot nodes --all
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All nodes rebooted
```

10. Update the Cloud Credential Operator **spec.credentialsMode** parameter to **Manual** by running the following command:

```
$ oc patch cloudcredential cluster \
  --type=merge \
  --patch '{"spec":{"credentialsMode":"Manual"}}'
```

11. Extract the list of **CredentialsRequest** objects from the OpenShift Container Platform release image by running the following command:

```
$ oc adm release extract \
  --credentials-requests \
  --included \
  --to <path_to_directory_for_credentials_requests> \
  --registry-config ~/.pull-secret
```



NOTE

This command might take a few moments to run.

12. Set an **AZURE_INSTALL_RG** variable with the Azure resource group name by running the following command:

```
$ AZURE_INSTALL_RG=`oc get infrastructure cluster -o jsonpath --template '{.status.platformStatus.azure.resourceGroupName}'`
```

13. Use the **ccoctl** utility to create managed identities for all **CredentialsRequest** objects by running the following command:

**NOTE**

The following command does not show all available options. For a complete list of options, including those that might be necessary for your specific use case, run **\$ ccoctl azure create-managed-identities --help**.

```
$ ccoctl azure create-managed-identities \
  --name <azure_infra_name> \
  --output-dir ./output_dir \
  --region <azure_region> \
  --subscription-id <azure_subscription_id> \
  --credentials-requests-dir <path_to_directory_for_credentials_requests> \
  --issuer-url "${OIDC_ISSUER_URL}" \
  --dnszone-resource-group-name <azure_dns_zone_resourcegroup_name> 1
  --installation-resource-group-name "${AZURE_INSTALL_RG}" \
  --network-resource-group-name <azure_resource_group> 2
```

- 1** Specify the name of the resource group that contains the DNS zone.
- 2** Optional: Specify the virtual network resource group if it is different from the cluster resource group.

14. Apply the Azure pod identity webhook configuration for Workload ID by running the following command:

```
$ oc apply -f ./output_dir/manifests/azure-ad-pod-identity-webhook-config.yaml
```

15. Apply the secrets generated by the **ccoctl** utility by running the following command:

```
$ find ./output_dir/manifests -iname "openshift*.yaml" -print0 | xargs -l {} -0 -t oc replace -f {}
```

This process might take several minutes.

16. Restart all of the pods in the cluster by running the following command:

```
$ oc adm reboot-machine-config-pool mcp/worker mcp/master
```

Restarting a pod updates the **serviceAccountIssuer** field and refreshes the service account public signing key.

17. Monitor the restart and update process by running the following command:

```
$ oc adm wait-for-node-reboot nodes --all
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All nodes rebooted
```

18. Monitor the configuration update progress by running the following command:

```
$ oc adm wait-for-stable-cluster
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All clusteroperators are stable
```

19. Optional: Remove the Azure root credentials secret by running the following command:

```
$ oc delete secret -n kube-system azure-credentials
```

10.4.3. Enabling AWS Security Token Service (STS) on an existing cluster

If you did not configure your Amazon Web Services (AWS) OpenShift Container Platform cluster to use Security Token Service (STS) during installation, you can enable this authentication method on an existing cluster.

IMPORTANT

The process to enable STS on an existing cluster is disruptive and takes a significant amount of time. Before proceeding, observe the following considerations:

- Read the following steps and ensure that you understand and accept the time requirement. The exact time requirement varies depending on the individual cluster, but it is likely to require at least one hour.
- During this process, you must refresh all service accounts and restart all pods on the cluster. These actions are disruptive to workloads. To mitigate this impact, you can temporarily halt these services and then redeploy them when the cluster is ready.
- Do not update the cluster until this process is complete.

Prerequisites

- You have installed an OpenShift Container Platform cluster on AWS.
- You have access to the cluster using an account with **cluster-admin** permissions.
- You have installed the OpenShift CLI (**oc**).
- You have extracted and prepared the Cloud Credential Operator utility (**ccoctl**) binary.
- You have access to your AWS account by using the AWS CLI (**aws**).

Procedure

1. Create an output directory for **ccoctl** generated manifests.

```
$ mkdir ./output_dir
```

2. Create the AWS Identity and Access Management (IAM) OpenID Connect (OIDC) provider.
 - a. Extract the service account public signing key for the cluster by running the following command:

```
$ oc get secret/next-bound-service-account-signing-key \
  -n openshift-kube-apiserver-operator \
  -ojsonpath='{ .data.service-account\.pub }' | base64 -d \
  > output_dir/serviceaccount-signer.public ❶
```

- ❶ This procedure uses a file named **serviceaccount-signer.public** as an example.

- b. Create the AWS IAM identity provider and S3 bucket by running the following command:

```
$ ./ccoctl aws create-identity-provider \
  --output-dir output_dir \ ❶
  --name <name_you_choose> \ ❷
  --region us-east-2 \ ❸
  --public-key-file output_dir/serviceaccount-signer.public ❹
```

- ❶ Specify the output directory you created earlier.
- ❷ Specify a globally unique name. This name functions as a prefix for AWS resources created by this command.
- ❸ Specify the AWS region of the cluster.
- ❹ Specify the relative path to the **serviceaccount-signer.public** file you created earlier.

- c. Save or note the Amazon Resource Name (ARN) for the IAM identity provider. You can find this information in the final line of the output of the previous command.

3. Update the cluster authentication configuration.

- a. Extract the OIDC issuer URL and update the authentication configuration of the cluster by running the following commands:

```
$ OIDC_ISSUER_URL=`awk '/serviceAccountIssuer/ { print $2 }'
  output_dir/manifests/cluster-authentication-02-config.yaml`
$ oc patch authentication cluster --type=merge -p '{"spec":
  {"serviceAccountIssuer":"'${OIDC_ISSUER_URL}'"}}
```

- b. Monitor the configuration update progress by running the following command:

```
$ oc adm wait-for-stable-cluster
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All clusteroperators are stable
```

4. Restart pods to apply the issuer update.

- a. Restart all of the pods in the cluster by running the following command:

```
$ oc adm reboot-machine-config-pool mcp/worker mcp/master
```


Restarting a pod updates the **serviceAccountIssuer** field and refreshes the service account public signing key.

- b. Monitor the restart and update process by running the following command:

```
$ oc adm wait-for-node-reboot nodes --all
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All nodes rebooted
```

5. Update the Cloud Credential Operator **spec.credentialsMode** parameter to **Manual** by running the following command:

```
$ oc patch cloudcredential cluster \
  --type=merge \
  --patch '{"spec":{"credentialsMode":"Manual"}}'
```

6. Extract **CredentialsRequests** objects.

- a. Create a **CLUSTER_VERSION** environment variable by running the following command:

```
$ CLUSTER_VERSION=$(oc get clusterversion version -o json | jq -r
  '.status.desired.version')
```

- b. Create a **CLUSTER_IMAGE** environment variable by running the following command:

```
$ CLUSTER_IMAGE=$(oc get clusterversion version -o json | jq -r ".status.history[] |
  select(.version == \"${CLUSTER_VERSION}\") | .image")
```

- c. Extract **CredentialsRequests** objects from the release image by running the following command:

```
$ oc adm release extract \
  --credentials-requests \
  --cloud=aws \
  --from ${CLUSTER_IMAGE} \
  --to output_dir/cred-reqs
```

7. Create AWS IAM roles and apply secrets.

- a. Create an IAM role for each **CredentialsRequests** object by running the following command:

```
$ ./ccoctl aws create-iam-roles \
  --output-dir ./output_dir/ \ ❶
  --name <name_you_choose> \ ❷
  --identity-provider-arn <identity_provider_arn> \ ❸
  --region us-east-2 \ ❹
  --credentials-requests-dir ./output_dir/cred-reqs/ ❺
```

- 1 Specify the output directory you created earlier.
- 2 Specify a globally unique name. This name functions as a prefix for AWS resources created by this command.
- 3 Specify the ARN for the IAM identity provider.
- 4 Specify the AWS region of the cluster.
- 5 Specify the relative path to the folder where you extracted the **CredentialsRequest** files with the **oc adm release extract** command.

b. Apply the generated secrets by running the following command:

```
$ find ./output_dir/manifests -iname "openshift*.yaml" -print0 | xargs -l {} -0 -t oc replace -f {}
```

8. Finish the configuration process by restarting the cluster.

a. Restart all of the pods in the cluster by running the following command:

```
$ oc adm reboot-machine-config-pool mcp/worker mcp/master
```

b. Monitor the restart and update process by running the following command:

```
$ oc adm wait-for-node-reboot nodes --all
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All nodes rebooted
```

c. Monitor the configuration update progress by running the following command:

```
$ oc adm wait-for-stable-cluster
```

This process might take 15 minutes or longer. The following output indicates that the process is complete:

```
All clusteroperators are stable
```

9. Optional: Remove the AWS root credentials secret by running the following command:

```
$ oc delete secret -n kube-system aws-creds
```

Additional resources

- [Microsoft Entra Workload ID](#)
- [Configuring an Azure cluster to use short-term credentials](#)
- [AWS Security Token Service](#)

- [Configuring an AWS cluster to use short-term credentials](#)

10.4.4. Verifying that a cluster uses short-term credentials

You can verify that a cluster uses short-term security credentials for individual components by checking the Cloud Credential Operator (CCO) configuration and other values in the cluster.

Prerequisites

- You deployed an OpenShift Container Platform cluster using the Cloud Credential Operator utility (**ccoctl**) to implement short-term credentials.
- You installed the OpenShift CLI (**oc**).
- You are logged in as a user with **cluster-admin** privileges.

Procedure

- Verify that the CCO is configured to operate in manual mode by running the following command:

```
$ oc get cloudcredentials cluster \
  -o=jsonpath={.spec.credentialsMode}
```

The following output confirms that the CCO is operating in manual mode:

Example output

```
Manual
```

- Verify that the cluster does not have **root** credentials by running the following command:

```
$ oc get secrets \
  -n kube-system <secret_name>
```

where **<secret_name>** is the name of the root secret for your cloud provider.

Platform	Secret name
Amazon Web Services (AWS)	aws-creds
Microsoft Azure	azure-credentials
Google Cloud	gcp-credentials

An error confirms that the root secret is not present on the cluster.

Example output for an AWS cluster

```
Error from server (NotFound): secrets "aws-creds" not found
```

- Verify that the components are using short-term security credentials for individual components by running the following command:

```
$ oc get authentication cluster \
  -o jsonpath \
  --template='{ .spec.serviceAccountIssuer }'
```

This command displays the value of the **.spec.serviceAccountIssuer** parameter in the cluster **Authentication** object. An output of a URL that is associated with your cloud provider indicates that the cluster is using manual mode with short-term credentials that are created and managed from outside of the cluster.

- Azure clusters: Verify that the components are assuming the Azure client ID that is specified in the secret manifests by running the following command:

```
$ oc get secrets \
  -n openshift-image-registry installer-cloud-credentials \
  -o jsonpath='{.data}'
```

An output that contains the **azure_client_id** and **azure_federated_token_file** fields confirms that the components are assuming the Azure client ID.

- Azure clusters: Verify that the pod identity webhook is running by running the following command:

```
$ oc get pods \
  -n openshift-cloud-credential-operator
```

Example output

NAME	READY	STATUS	RESTARTS	AGE
cloud-credential-operator-59cf744f78-r8pbq	2/2	Running	2	71m
pod-identity-webhook-548f977b4c-859lz	1/1	Running	1	70m

10.5. ADDITIONAL RESOURCES

- [About the Cloud Credential Operator](#)

CHAPTER 11. CONFIGURING ALERT NOTIFICATIONS

In OpenShift Container Platform, an alert is fired when the conditions defined in an alerting rule are true. An alert provides a notification that a set of circumstances are apparent within a cluster. Firing alerts can be viewed in the Alerting UI in the OpenShift Container Platform web console by default. After an installation, you can configure OpenShift Container Platform to send alert notifications to external systems.

11.1. SENDING NOTIFICATIONS TO EXTERNAL SYSTEMS

In OpenShift Container Platform, 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
- Microsoft Teams

Routing alerts to receivers enables you to send timely notifications to the appropriate teams when failures occur. For example, critical alerts require immediate attention and are typically paged to an individual or a critical response team. Alerts that provide non-critical warning notifications might instead be routed to a ticketing system for non-immediate review.

Checking that alerting is operational by using the watchdog alert

OpenShift Container Platform monitoring includes a watchdog alert that fires continuously. Alertmanager repeatedly sends watchdog alert notifications to configured notification providers. The provider is usually configured to notify an administrator when it stops receiving the watchdog alert. This mechanism helps you quickly identify any communication issues between Alertmanager and the notification provider.

11.2. ADDITIONAL RESOURCES

- [About OpenShift Container Platform monitoring](#)
- [Configuring alerts and notifications for core platform monitoring](#)
- [Configuring alerts and notifications for user workload monitoring](#)

CHAPTER 12. CONVERTING A CONNECTED CLUSTER TO A DISCONNECTED CLUSTER

There might be some scenarios where you need to convert your OpenShift Container Platform cluster from a connected cluster to a disconnected cluster.

A disconnected cluster, also known as a restricted cluster, does not have an active connection to the internet. As such, you must mirror the contents of your registries and installation media. You can create this mirror registry on a host that can access both the internet and your closed network, or copy images to a device that you can move across network boundaries.

For information on how to convert your cluster, see the [Converting a connected cluster to a disconnected cluster](#) procedure in the Disconnected environments section.