



OpenShift Container Platform 4.20

Nodes

Configuring and managing nodes in OpenShift Container Platform

OpenShift Container Platform 4.20 Nodes

Configuring and managing nodes in 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 for configuring and managing the nodes, Pods, and containers in your cluster. It also provides information on configuring Pod scheduling and placement, using jobs and DaemonSets to automate tasks, and other tasks to ensure an efficient cluster.

Table of Contents

CHAPTER 1. OVERVIEW OF NODES	14
1.1. ABOUT NODES	14
1.1.1. Read operations	15
1.1.2. Enhancement operations	15
1.2. ABOUT PODS	15
1.2.1. Read operations	15
1.2.2. Management operations	15
1.2.3. Enhancement operations	16
1.3. ABOUT CONTAINERS	17
1.4. ABOUT AUTOSCALING PODS ON A NODE	17
1.5. GLOSSARY OF COMMON TERMS FOR OPENSIFT CONTAINER PLATFORM NODES	18
CHAPTER 2. WORKING WITH PODS	20
2.1. USING PODS	20
2.1.1. Understanding pods	20
2.1.2. Example pod configurations	20
2.1.3. Understanding resource requests and limits	22
2.1.4. Additional resources	23
2.2. VIEWING PODS	23
2.2.1. Viewing pods in a project	23
2.2.2. Viewing pod usage statistics	24
2.2.3. Viewing resource logs	24
2.2.3.1. Viewing resource logs by using the web console	24
2.2.3.2. Viewing resource logs by using the CLI	25
2.3. CONFIGURING AN OPENSIFT CONTAINER PLATFORM CLUSTER FOR PODS	26
2.3.1. Configuring how pods behave after restart	26
2.3.2. Limiting the bandwidth available to pods	27
2.3.3. Understanding how to use pod disruption budgets to specify the number of pods that must be up	28
2.3.3.1. Specifying the number of pods that must be up with pod disruption budgets	29
2.3.3.2. Specifying the eviction policy for unhealthy pods	30
2.3.4. Preventing pod removal using critical pods	31
2.3.5. Reducing pod timeouts when using persistent volumes with high file counts	32
2.4. AUTOMATICALLY SCALING PODS WITH THE HORIZONTAL POD AUTOSCALER	32
2.4.1. Understanding horizontal pod autoscalers	32
2.4.2. How does the HPA work?	34
2.4.3. About requests and limits	35
2.4.4. Best practices	36
2.4.4.1. Scaling policies	36
2.4.5. Creating a horizontal pod autoscaler by using the web console	39
2.4.5.1. Editing a horizontal pod autoscaler by using the web console	40
2.4.5.2. Removing a horizontal pod autoscaler by using the web console	40
2.4.6. Creating a horizontal pod autoscaler by using the CLI	40
2.4.6.1. Creating a horizontal pod autoscaler for a percent of CPU use	40
2.4.6.2. Creating a horizontal pod autoscaler for a specific CPU value	42
2.4.6.3. Creating a horizontal pod autoscaler object for a percent of memory use	44
2.4.6.4. Creating a horizontal pod autoscaler object for specific memory use	48
2.4.7. Understanding horizontal pod autoscaler status conditions by using the CLI	51
2.4.7.1. Viewing horizontal pod autoscaler status conditions by using the CLI	53
2.4.8. Additional resources	54
2.5. AUTOMATICALLY ADJUST POD RESOURCE LEVELS WITH THE VERTICAL POD AUTOSCALER	54
2.5.1. About the Vertical Pod Autoscaler Operator	55

2.5.2. Installing the Vertical Pod Autoscaler Operator	56
2.5.3. Moving the Vertical Pod Autoscaler Operator components	57
2.5.4. About using the Vertical Pod Autoscaler Operator	62
2.5.4.1. Changing the VPA minimum value	64
2.5.4.2. Automatically applying VPA recommendations	64
2.5.4.3. Automatically applying VPA recommendations on pod creation	66
2.5.4.4. Manually applying VPA recommendations	66
2.5.4.5. Exempting containers from applying VPA recommendations	67
2.5.4.6. Performance tuning the VPA Operator	69
2.5.4.7. Custom memory bump-up after OOM event	75
2.5.4.8. Using an alternative recommender	75
2.5.5. Using the Vertical Pod Autoscaler Operator	78
2.5.5.1. Example custom resources for the Vertical Pod Autoscaler	81
2.5.6. Uninstalling the Vertical Pod Autoscaler Operator	83
2.6. ADJUST POD RESOURCE LEVELS WITHOUT POD DISRUPTION	84
2.6.1. About in-place pod resizing	84
2.6.2. Configuring in-place pod resizing	86
2.6.3. Additional resources	88
2.7. PROVIDING SENSITIVE DATA TO PODS BY USING SECRETS	88
2.7.1. Understanding secrets	88
2.7.1.1. Types of secrets	89
2.7.1.2. Secret data keys	89
2.7.1.3. Automatically generated image pull secrets	90
2.7.2. Understanding how to create secrets	90
2.7.2.1. Secret creation restrictions	93
2.7.2.2. Creating an opaque secret	93
2.7.2.3. Creating a legacy service account token secret	94
2.7.2.4. Creating a basic authentication secret	95
2.7.2.5. Creating an SSH authentication secret	96
2.7.2.6. Creating a Docker configuration secret	97
2.7.2.7. Creating a secret using the web console	98
2.7.3. Understanding how to update secrets	99
2.7.4. Creating and using secrets	99
2.7.5. About using signed certificates with secrets	100
2.7.5.1. Generating signed certificates for use with secrets	101
2.7.6. Troubleshooting secrets	103
2.8. PROVIDING SENSITIVE DATA TO PODS BY USING AN EXTERNAL SECRETS STORE	104
2.8.1. About the Secrets Store CSI Driver Operator	104
2.8.1.1. Secrets store providers	104
2.8.1.2. Automatic rotation	105
2.8.2. Installing the Secrets Store CSI driver	105
2.8.3. Mounting secrets from an external secrets store to a CSI volume	106
2.8.3.1. Mounting secrets from AWS Secrets Manager	106
2.8.3.2. Mounting secrets from AWS Systems Manager Parameter Store	112
2.8.3.3. Mounting secrets from Azure Key Vault	118
2.8.3.4. Mounting secrets from Google Secret Manager	123
2.8.3.5. Mounting secrets from HashiCorp Vault	128
2.8.4. Enabling synchronization of mounted content as Kubernetes secrets	134
2.8.5. Viewing the status of secrets in the pod volume mount	136
2.8.6. Uninstalling the Secrets Store CSI Driver Operator	137
2.9. AUTHENTICATING PODS WITH SHORT-TERM CREDENTIALS	138
2.9.1. Configuring short-term authentication for workloads	138
2.9.1.1. Environment and user access requirements	138

2.9.2. Configuring GCP Workload Identity authentication for applications on Google Cloud	138
2.9.2.1. Creating a federated Google Cloud service account	139
2.9.2.2. Creating an OpenShift Container Platform service account for Google Cloud	139
2.9.2.3. Deploying customer workloads that authenticate with GCP Workload Identity	141
2.10. CREATING AND USING CONFIG MAPS	144
2.10.1. Understanding config maps	144
2.10.1.1. Config map restrictions	145
2.10.2. Creating a config map in the OpenShift Container Platform web console	145
2.10.3. Creating a config map by using the CLI	145
2.10.3.1. Creating a config map from a directory	146
2.10.3.2. Creating a config map from a file	148
2.10.3.3. Creating a config map from literal values	150
2.10.4. Use cases: Consuming config maps in pods	151
2.10.4.1. Populating environment variables in containers by using config maps	151
2.10.4.2. Setting command-line arguments for container commands with config maps	153
2.10.4.3. Injecting content into a volume by using config maps	154
2.11. MOUNTING AN OCI IMAGE INTO A POD	155
2.11.1. Understanding image volumes	156
2.11.2. Adding an image volume to a pod	156
2.12. USING DEVICE PLUGINS TO ACCESS EXTERNAL RESOURCES WITH PODS	158
2.12.1. Understanding device plugins	158
2.12.1.1. Example device plugins	158
2.12.1.2. Methods for deploying a device plugin	159
2.12.2. Understanding the Device Manager	159
2.12.3. Enabling Device Manager	160
2.13. INCLUDING POD PRIORITY IN POD SCHEDULING DECISIONS	161
2.13.1. Understanding pod priority	161
2.13.1.1. Pod priority classes	161
2.13.1.2. Pod priority names	162
2.13.2. Understanding pod preemption	162
2.13.2.1. Non-preempting priority classes	163
2.13.2.2. Pod preemption and other scheduler settings	163
2.13.2.3. Graceful termination of preempted pods	164
2.13.3. Configuring priority and preemption	164
2.14. PLACING PODS ON SPECIFIC NODES USING NODE SELECTORS	165
2.14.1. Using node selectors to control pod placement	165
2.15. ALLOCATING GPUS TO PODS	169
2.15.1. About allocating GPUs to workloads	169
2.15.2. About GPU allocation objects	170
2.15.3. Adding resource claims to pods	172
2.16. RUN ONCE DURATION OVERRIDE OPERATOR	174
2.16.1. Run Once Duration Override Operator overview	174
2.16.1.1. About the Run Once Duration Override Operator	174
2.16.2. Run Once Duration Override Operator release notes	175
2.16.2.1. Run Once Duration Override Operator 1.3.1	175
2.16.2.1.1. New features and enhancements	175
2.16.2.1.2. Bug fixes	175
2.16.2.2. Run Once Duration Override Operator 1.3.0	175
2.16.2.2.1. Bug fixes	175
2.16.3. Overriding the active deadline for run-once pods	175
2.16.3.1. Installing the Run Once Duration Override Operator	176
2.16.3.2. Enabling the run-once duration override on a namespace	177
2.16.3.3. Updating the run-once active deadline override value	178

2.16.4. Uninstalling the Run Once Duration Override Operator	179
2.16.4.1. Uninstalling the Run Once Duration Override Operator	179
2.16.4.2. Uninstalling Run Once Duration Override Operator resources	180
2.17. RUNNING PODS IN LINUX USER NAMESPACES	181
2.17.1. Configuring Linux user namespace support	181

CHAPTER 3. AUTOMATICALLY SCALING PODS WITH THE CUSTOM METRICS AUTOSCALER OPERATOR . 187

3.1. RELEASE NOTES	187
3.1.1. Custom Metrics Autoscaler Operator release notes	187
3.1.1.1. Supported versions	187
3.1.1.2. Custom Metrics Autoscaler Operator 2.17.2-2 release notes	187
3.1.2. Release notes for past releases of the Custom Metrics Autoscaler Operator	188
3.1.2.1. Custom Metrics Autoscaler Operator 2.17.2 release notes	188
3.1.2.1.1. New features and enhancements	188
3.1.2.1.1.1. The KEDA controller is automatically created during installation	188
3.1.2.1.1.2. Support for the Kubernetes workload trigger	188
3.1.2.1.1.3. Support for bound service account tokens	188
3.1.2.1.2. Bug fixes	188
3.1.2.2. Custom Metrics Autoscaler Operator 2.15.1-4 release notes	189
3.1.2.2.1. New features and enhancements	189
3.1.2.2.1.1. CMA multi-arch builds	189
3.1.2.3. Custom Metrics Autoscaler Operator 2.14.1-467 release notes	189
3.1.2.3.1. Bug fixes	189
3.1.2.4. Custom Metrics Autoscaler Operator 2.14.1-454 release notes	190
3.1.2.4.1. New features and enhancements	190
3.1.2.4.1.1. Support for the Cron trigger with the Custom Metrics Autoscaler Operator	190
3.1.2.4.2. Bug fixes	190
3.1.2.5. Custom Metrics Autoscaler Operator 2.13.1 release notes	190
3.1.2.5.1. New features and enhancements	190
3.1.2.5.1.1. Support for custom certificates with the Custom Metrics Autoscaler Operator	190
3.1.2.5.2. Bug fixes	191
3.1.2.6. Custom Metrics Autoscaler Operator 2.12.1-394 release notes	191
3.1.2.6.1. Bug fixes	191
3.1.2.7. Custom Metrics Autoscaler Operator 2.12.1-384 release notes	192
3.1.2.7.1. Bug fixes	192
3.1.2.8. Custom Metrics Autoscaler Operator 2.12.1-376 release notes	192
3.1.2.8.1. Bug fixes	192
3.1.2.9. Custom Metrics Autoscaler Operator 2.11.2-322 release notes	193
3.1.2.9.1. Bug fixes	193
3.1.2.10. Custom Metrics Autoscaler Operator 2.11.2-311 release notes	193
3.1.2.10.1. New features and enhancements	193
3.1.2.10.1.1. Red Hat OpenShift Service on AWS and OpenShift Dedicated are now supported	193
3.1.2.10.2. Bug fixes	193
3.1.2.11. Custom Metrics Autoscaler Operator 2.10.1-267 release notes	194
3.1.2.11.1. Bug fixes	194
3.1.2.12. Custom Metrics Autoscaler Operator 2.10.1 release notes	194
3.1.2.12.1. New features and enhancements	195
3.1.2.12.1.1. Custom Metrics Autoscaler Operator general availability	195
3.1.2.12.1.2. Performance metrics	195
3.1.2.12.1.3. Pausing the custom metrics autoscaling for scaled objects	195
3.1.2.12.1.4. Replica fall back for scaled objects	195
3.1.2.12.1.5. Customizable HPA naming for scaled objects	195

3.1.2.12.1.6. Activation and scaling thresholds	195
3.1.2.13. Custom Metrics Autoscaler Operator 2.8.2-174 release notes	195
3.1.2.13.1. New features and enhancements	196
3.1.2.13.1.1. Operator upgrade support	196
3.1.2.13.1.2. must-gather support	196
3.1.2.14. Custom Metrics Autoscaler Operator 2.8.2 release notes	196
3.1.2.14.1. New features and enhancements	196
3.1.2.14.1.1. Audit Logging	196
3.1.2.14.1.2. Scale applications based on Apache Kafka metrics	196
3.1.2.14.1.3. Scale applications based on CPU metrics	196
3.1.2.14.1.4. Scale applications based on memory metrics	196
3.2. CUSTOM METRICS AUTOSCALER OPERATOR OVERVIEW	196
3.2.1. Custom CA certificates for the Custom Metrics Autoscaler	199
3.3. INSTALLING THE CUSTOM METRICS AUTOSCALER	199
3.3.1. Installing the custom metrics autoscaler	199
3.3.2. Editing the Keda Controller CR	201
3.4. UNDERSTANDING CUSTOM METRICS AUTOSCALER TRIGGERS	202
3.4.1. Understanding the Prometheus trigger	202
3.4.1.1. Configuring GPU-based autoscaling with Prometheus and DCGM metrics	204
3.4.1.2. Configuring the custom metrics autoscaler to use OpenShift Container Platform monitoring	204
3.4.2. Understanding the CPU trigger	208
3.4.3. Understanding the memory trigger	209
3.4.4. Understanding the Kafka trigger	210
3.4.5. Understanding the Cron trigger	212
3.4.6. Understanding the Kubernetes workload trigger	212
3.5. UNDERSTANDING CUSTOM METRICS AUTOSCALER TRIGGER AUTHENTICATIONS	213
3.5.1. Using trigger authentications	218
3.6. UNDERSTANDING HOW TO ADD CUSTOM METRICS AUTOSCALERS	221
3.6.1. Adding a custom metrics autoscaler to a workload	221
3.6.2. Adding a custom metrics autoscaler to a job	225
3.6.3. Additional resources	228
3.7. PAUSING THE CUSTOM METRICS AUTOSCALER FOR A SCALED OBJECT	228
3.7.1. Pausing a custom metrics autoscaler	228
3.7.2. Restarting the custom metrics autoscaler for a scaled object	229
3.8. GATHERING AUDIT LOGS	230
3.8.1. Configuring audit logging	230
3.9. GATHERING DEBUGGING DATA	233
3.9.1. Gathering debugging data	233
3.10. VIEWING OPERATOR METRICS	236
3.10.1. Accessing performance metrics	236
3.10.1.1. Provided Operator metrics	237
3.11. REMOVING THE CUSTOM METRICS AUTOSCALER OPERATOR	237
3.11.1. Uninstalling the Custom Metrics Autoscaler Operator	238
CHAPTER 4. CONTROLLING POD PLACEMENT ONTO NODES (SCHEDULING)	240
4.1. CONTROLLING POD PLACEMENT USING THE SCHEDULER	240
4.1.1. About the default scheduler	240
4.1.1.1. Understanding default scheduling	240
4.1.2. Scheduler use cases	241
4.1.2.1. Infrastructure topological levels	241
4.1.2.2. Affinity	241
4.1.2.3. Anti-affinity	241
4.2. SCHEDULING PODS USING A SCHEDULER PROFILE	241

4.2.1. About scheduler profiles	242
4.2.2. Configuring a scheduler profile	242
4.3. PLACING PODS RELATIVE TO OTHER PODS USING AFFINITY AND ANTI-AFFINITY RULES	243
4.3.1. Understanding pod affinity	243
4.3.2. Configuring a pod affinity rule	245
4.3.3. Configuring a pod anti-affinity rule	247
4.3.4. Sample pod affinity and anti-affinity rules	248
4.3.4.1. Pod Affinity	248
4.3.4.2. Pod Anti-affinity	249
4.3.4.3. Pod Affinity with no Matching Labels	251
4.3.5. Using pod affinity and anti-affinity to control where an Operator is installed	252
4.4. CONTROLLING POD PLACEMENT ON NODES USING NODE AFFINITY RULES	254
4.4.1. Understanding node affinity	254
4.4.2. Configuring a required node affinity rule	257
4.4.3. Configuring a preferred node affinity rule	258
4.4.4. Sample node affinity rules	259
4.4.4.1. Node affinity with matching labels	259
4.4.4.2. Node affinity with no matching labels	261
4.4.5. Using node affinity to control where an Operator is installed	262
4.4.6. Additional resources	264
4.5. PLACING PODS ONTO OVERCOMMITTED NODES	264
4.5.1. Understanding overcommitment	264
4.5.2. Understanding nodes overcommitment	265
4.6. CONTROLLING POD PLACEMENT USING NODE TAINTS	266
4.6.1. Understanding taints and tolerations	266
4.6.1.1. Understanding how to use toleration seconds to delay pod evictions	269
4.6.1.2. Understanding how to use multiple taints	269
4.6.1.3. Understanding pod scheduling and node conditions (taint node by condition)	270
4.6.1.4. Understanding evicting pods by condition (taint-based evictions)	271
4.6.1.5. Tolerating all taints	272
4.6.2. Adding taints and tolerations	272
4.6.2.1. Adding taints and tolerations using a compute machine set	274
4.6.2.2. Binding a user to a node using taints and tolerations	276
4.6.2.3. Creating a project with a node selector and toleration	277
4.6.2.4. Controlling nodes with special hardware using taints and tolerations	278
4.6.3. Removing taints and tolerations	279
4.7. PLACING PODS ON SPECIFIC NODES USING NODE SELECTORS	280
4.7.1. About node selectors	280
4.7.2. Using node selectors to control pod placement	284
4.7.3. Creating default cluster-wide node selectors	288
4.7.4. Creating project-wide node selectors	291
4.8. CONTROLLING POD PLACEMENT BY USING POD TOPOLOGY SPREAD CONSTRAINTS	295
4.8.1. Example use cases	295
4.8.2. Important considerations	295
4.8.3. Understanding skew and maxSkew	295
4.8.3.1. Example skew calculation	295
4.8.3.2. The maxSkew parameter	296
4.8.4. Example configurations for pod topology spread constraints	296
4.8.5. Additional resources	298
4.9. DESCHEDULER	298
4.9.1. Descheduler overview	298
4.9.1.1. About the descheduler	298
4.9.1.2. Descheduler profiles	299

4.9.2. Kube Descheduler Operator release notes	303
4.9.2.1. Release notes for Kube Descheduler Operator 5.3.1	304
4.9.2.1.1. New features and enhancements	304
4.9.2.2. Release notes for Kube Descheduler Operator 5.3.0	304
4.9.2.2.1. New features and enhancements	304
4.9.2.3. Release notes for Kube Descheduler Operator 5.2.1	304
4.9.2.3.1. New features and enhancements	304
4.9.2.4. Release notes for Kube Descheduler Operator 5.2.0	304
4.9.2.4.1. New features and enhancements	305
4.9.2.4.2. Bug fixes	305
4.9.3. Evicting pods using the descheduler	305
4.9.3.1. Installing the descheduler	305
4.9.3.2. Configuring descheduler profiles	307
4.9.3.3. Configuring the descheduler interval	308
4.9.4. Uninstalling the Kube Descheduler Operator	309
4.9.4.1. Uninstalling the descheduler	309
4.10. SECONDARY SCHEDULER	310
4.10.1. Secondary scheduler overview	310
4.10.1.1. About the Secondary Scheduler Operator	310
4.10.2. Secondary Scheduler Operator for Red Hat OpenShift release notes	311
4.10.2.1. Release notes for Secondary Scheduler Operator for Red Hat OpenShift 1.5.0	311
4.10.2.1.1. New features and enhancements	311
4.10.2.1.2. Known issues	311
4.10.3. Scheduling pods using a secondary scheduler	311
4.10.3.1. Installing the Secondary Scheduler Operator	311
4.10.3.2. Deploying a secondary scheduler	312
4.10.3.3. Scheduling a pod using the secondary scheduler	314
4.10.4. Uninstalling the Secondary Scheduler Operator	316
4.10.4.1. Uninstalling the Secondary Scheduler Operator	316
4.10.4.2. Removing Secondary Scheduler Operator resources	316
CHAPTER 5. USING JOBS AND DAEMON SETS	318
5.1. RUNNING BACKGROUND TASKS ON NODES AUTOMATICALLY WITH DAEMON SETS	318
5.1.1. Scheduled by default scheduler	318
5.1.2. Creating daemonsets	319
5.2. RUNNING TASKS IN PODS USING JOBS	321
5.2.1. Understanding jobs and cron jobs	322
5.2.1.1. Understanding how to create jobs	323
5.2.1.2. Understanding how to set a maximum duration for jobs	323
5.2.1.3. Understanding how to set a job back off policy for pod failure	324
5.2.1.4. Understanding how to configure a cron job to remove artifacts	324
5.2.1.5. Known limitations	324
5.2.2. Creating jobs	324
5.2.3. Creating cron jobs	326
CHAPTER 6. WORKING WITH NODES	328
6.1. VIEWING AND LISTING THE NODES IN YOUR OPENSIFT CONTAINER PLATFORM CLUSTER	328
6.1.1. About listing all the nodes in a cluster	328
6.1.2. Listing pods on a node in your cluster	333
6.1.3. Viewing memory and CPU usage statistics on your nodes	333
6.2. WORKING WITH NODES	334
6.2.1. Understanding how to evacuate pods on nodes	334
6.2.2. Understanding how to update labels on nodes	335

6.2.3. Understanding how to mark nodes as unschedulable or schedulable	336
6.2.4. Handling errors in single-node OpenShift clusters when the node reboots without draining application pods	337
6.2.5. Deleting nodes	338
6.2.5.1. Deleting nodes from a cluster	338
6.2.5.2. Deleting nodes from a bare metal cluster	339
6.3. MANAGING NODES	339
6.3.1. Modifying nodes	340
6.3.2. Configuring control plane nodes as schedulable	341
6.3.3. Setting SELinux booleans	342
6.3.4. Adding kernel arguments to nodes	343
6.3.5. Allowing swap memory use on nodes	347
6.3.6. About configuring parallel container image pulls	349
6.3.6.1. Configuring parallel container image pulls	349
6.3.7. Migrating control plane nodes from one RHOSP host to another manually	351
6.3.8. Additional resources	353
6.4. ADDING WORKER NODES TO AN ON-PREMISE CLUSTER	353
6.4.1. Supported platforms	353
6.4.2. Supported architectures	354
6.4.3. Adding nodes to your cluster	354
6.4.3.1. Adding one or more nodes using a configuration file	354
6.4.3.2. Adding a node with command flags	356
6.4.4. Cluster configuration reference	357
6.4.4.1. YAML file parameters	357
6.4.4.2. Command flag options	359
6.5. MANAGING THE MAXIMUM NUMBER OF PODS PER NODE	362
6.5.1. Configuring the maximum number of pods per node	363
6.6. REPLACING A FAILED BARE-METAL CONTROL PLANE NODE WITHOUT BMC CREDENTIALS	365
6.6.1. Prerequisites	365
6.6.2. Removing the unhealthy etcd member	365
6.6.3. Deleting the machine of the unhealthy etcd member	368
6.6.4. Verifying that the failed node was deleted	370
6.6.5. Creating the new control plane node	371
6.6.6. Linking the node, bare metal host, and machine together	373
6.6.7. Adding the new etcd member	376
6.6.8. Additional resources	377
6.7. USING THE NODE TUNING OPERATOR	378
6.7.1. Accessing an example Node Tuning Operator specification	378
6.7.2. Custom tuning specification	379
6.7.3. Default profiles set on a cluster	384
6.7.4. Supported TuneD daemon plugins	384
6.8. REMEDIATING, FENCING, AND MAINTAINING NODES	385
6.9. UNDERSTANDING NODE REBOOTING	385
6.9.1. About rebooting nodes running critical infrastructure	386
6.9.2. Rebooting a node using pod anti-affinity	386
6.9.3. Understanding how to reboot nodes running routers	387
6.9.4. Rebooting a node gracefully	387
6.10. BOOT IMAGE MANAGEMENT	389
6.10.1. About boot image management	390
6.10.2. Disabling boot image management	393
6.10.3. Enabling boot image management	396
6.11. FREEING NODE RESOURCES USING GARBAGE COLLECTION	400
6.11.1. Understanding how terminated containers are removed through garbage collection	400

6.11.2. Understanding how images are removed through garbage collection	401
6.11.3. Configuring garbage collection for containers and images	402
6.12. ALLOCATING RESOURCES FOR NODES IN AN OPENSIFT CONTAINER PLATFORM CLUSTER	405
6.12.1. Understanding how to allocate resources for nodes	405
6.12.1.1. How OpenShift Container Platform computes allocated resources	406
6.12.1.2. How nodes enforce resource constraints	406
6.12.1.3. Understanding Eviction Thresholds	407
6.12.1.4. How the scheduler determines resource availability	407
6.12.2. Understanding process ID limits	408
6.12.2.1. Risks of setting higher process ID limits for OpenShift Container Platform pods	409
6.12.3. Automatically allocating resources for nodes	409
6.12.4. Manually allocating resources for nodes	411
6.13. ALLOCATING SPECIFIC CPUS FOR NODES IN A CLUSTER	413
6.13.1. Reserving CPUs for nodes	413
6.14. ENABLING TLS SECURITY PROFILES FOR THE KUBELET	414
6.14.1. Understanding TLS security profiles	414
6.14.2. Configuring the TLS security profile for the kubelet	415
6.15. CREATING INFRASTRUCTURE NODES	418
6.15.1. OpenShift Container Platform infrastructure components	418
6.15.1.1. Creating an infrastructure node	419
CHAPTER 7. WORKING WITH CONTAINERS	421
7.1. UNDERSTANDING CONTAINERS	421
7.1.1. About containers and RHEL kernel memory	421
7.1.2. About the container engine and container runtime	421
7.2. USING INIT CONTAINERS TO PERFORM TASKS BEFORE A POD IS DEPLOYED	422
7.2.1. Understanding Init Containers	422
7.2.2. Creating Init Containers	423
7.3. USING VOLUMES TO PERSIST CONTAINER DATA	425
7.3.1. Understanding volumes	425
7.3.2. Working with volumes using the OpenShift Container Platform CLI	426
7.3.3. Listing volumes and volume mounts in a pod	427
7.3.4. Adding volumes to a pod	427
7.3.5. Updating volumes and volume mounts in a pod	432
7.3.6. Removing volumes and volume mounts from a pod	434
7.3.7. Configuring volumes for multiple uses in a pod	435
7.4. MAPPING VOLUMES USING PROJECTED VOLUMES	436
7.4.1. Understanding projected volumes	437
7.4.1.1. Example Pod specs	438
7.4.1.2. Pathing Considerations	440
7.4.2. Configuring a Projected Volume for a Pod	441
7.5. ALLOWING CONTAINERS TO CONSUME API OBJECTS	444
7.5.1. Expose pod information to Containers using the Downward API	444
7.5.2. Understanding how to consume container values using the downward API	445
7.5.2.1. Consuming container values using environment variables	445
7.5.2.2. Consuming container values using a volume plugin	447
7.5.3. Understanding how to consume container resources using the Downward API	448
7.5.3.1. Consuming container resources using environment variables	448
7.5.3.2. Consuming container resources using a volume plugin	449
7.5.4. Consuming secrets using the Downward API	451
7.5.5. Consuming configuration maps using the Downward API	452
7.5.6. Referencing environment variables	453
7.5.7. Escaping environment variable references	454

7.6. COPYING FILES TO OR FROM OPENSIFT CONTAINER PLATFORM CONTAINERS	455
7.6.1. Understanding how to copy files	455
7.6.1.1. Requirements	455
7.6.2. Copying files to and from containers	456
7.6.3. Using advanced Rsync features	456
7.7. EXECUTING REMOTE COMMANDS IN AN OPENSIFT CONTAINER PLATFORM CONTAINER	457
7.7.1. Executing remote commands in containers	457
7.7.2. Protocol for initiating a remote command from a client	457
7.8. USING PORT FORWARDING TO ACCESS APPLICATIONS IN A CONTAINER	458
7.8.1. Understanding port forwarding	458
7.8.2. Using port forwarding	459
7.8.3. Protocol for initiating port forwarding from a client	460
7.9. USING SYSCTLs IN CONTAINERS	460
7.9.1. About sysctls	461
7.9.2. Namespaced and node-level sysctls	461
7.9.3. Safe and unsafe sysctls	461
7.9.4. Updating the interface-specific safe sysctls list	464
7.9.5. Starting a pod with safe sysctls	468
7.9.6. Starting a pod with unsafe sysctls	470
7.9.7. Enabling unsafe sysctls	471
7.9.8. Additional resources	474
7.10. ACCESSING FASTER BUILDS WITH /DEV/FUSE	474
7.10.1. Configuring /dev/fuse for unprivileged builds in pods	474
CHAPTER 8. WORKING WITH CLUSTERS	478
8.1. VIEWING SYSTEM EVENT INFORMATION IN OPENSIFT CONTAINER PLATFORM CLUSTERS	478
8.1.1. Understanding events	478
8.1.2. Viewing events using the CLI	478
8.1.3. List of events	479
8.2. ESTIMATING THE NUMBER OF PODS YOUR OPENSIFT CONTAINER PLATFORM NODES CAN HOLD	487
8.2.1. Understanding the OpenShift Cluster Capacity Tool	487
8.2.2. Running the OpenShift Cluster Capacity Tool on the command line	488
8.2.3. Running the OpenShift Cluster Capacity Tool as a job inside a pod	490
8.2.4. Additional resources	493
8.3. RESTRICT RESOURCE CONSUMPTION WITH LIMIT RANGES	493
8.3.1. About limit ranges	493
8.3.2. About component limits	494
8.3.3. Creating a Limit Range	499
8.3.4. Viewing a limit	501
8.3.5. Deleting a Limit Range	502
8.4. CONFIGURING CLUSTER MEMORY TO MEET CONTAINER MEMORY AND RISK REQUIREMENTS	502
8.4.1. Understanding how to manage application memory	502
8.4.2. Understanding OpenJDK settings for OpenShift Container Platform	504
8.4.3. Finding the memory request and limit from within a pod	505
8.4.4. Understanding OOM kill policy	507
8.4.5. Understanding pod eviction	509
8.5. CONFIGURING YOUR CLUSTER TO PLACE PODS ON OVERCOMMITTED NODES	509
8.5.1. Resource requests and overcommitment	510
8.5.2. Cluster-level overcommit using the Cluster Resource Override Operator	510
8.5.2.1. Installing the Cluster Resource Override Operator using the web console	512
8.5.2.2. Installing the Cluster Resource Override Operator using the CLI	515
8.5.2.3. Configuring cluster-level overcommit	518

8.5.2.4. Moving the Cluster Resource Override Operator pods	519
8.5.3. Node-level overcommit	523
8.5.3.1. Understanding container CPU and memory requests	523
8.5.3.2. Understanding overcommitment and quality of service classes	523
8.5.3.2.1. Understanding how to reserve memory across quality of service tiers	524
8.5.3.3. Understanding swap memory and QoS	524
8.5.3.4. Understanding nodes overcommitment	525
8.5.3.5. Disabling or enforcing CPU limits using CPU CFS quotas	526
8.5.3.6. Reserving resources for system processes	527
8.5.3.7. Disabling overcommitment for a node	527
8.5.4. Project-level limits	527
8.5.4.1. Disabling overcommitment for a project	527
8.5.5. Additional resources	528
8.6. ENABLING FEATURES USING FEATURE GATES	528
8.6.1. Understanding feature gates	528
8.6.2. Enabling feature sets at installation	532
8.6.3. Enabling feature sets using the web console	533
8.6.4. Enabling feature sets using the CLI	535
8.6.5. Additional resources	536
8.7. IMPROVING CLUSTER STABILITY IN HIGH LATENCY ENVIRONMENTS USING WORKER LATENCY PROFILES	537
8.7.1. Understanding worker latency profiles	538
8.7.2. Using and changing worker latency profiles	540
CHAPTER 9. REMOTE WORKER NODES ON THE NETWORK EDGE	544
9.1. USING REMOTE WORKER NODES AT THE NETWORK EDGE	544
9.1.1. Adding remote worker nodes	544
9.1.2. Network separation with remote worker nodes	545
9.1.3. Power loss on remote worker nodes	546
9.1.4. Latency spikes or temporary reduction in throughput to remote workers	546
9.1.5. Remote worker node strategies	547
CHAPTER 10. WORKER NODES FOR SINGLE-NODE OPENSIFT CLUSTERS	552
10.1. ADDING WORKER NODES TO SINGLE-NODE OPENSIFT CLUSTERS	552
10.1.1. Requirements for installing single-node OpenShift worker nodes	552
10.1.2. Adding worker nodes using the Assisted Installer and OpenShift Cluster Manager	554
10.1.3. Adding worker nodes using the Assisted Installer API	554
10.1.3.1. Authenticating against the Assisted Installer REST API	555
10.1.3.2. Adding worker nodes using the Assisted Installer REST API	556
10.1.4. Adding worker nodes to single-node OpenShift clusters manually	562
10.1.5. Approving the certificate signing requests for your machines	566
CHAPTER 11. NODE METRICS DASHBOARD	569
11.1. ABOUT THE NODE METRICS DASHBOARD	569
11.2. ACCESSING THE NODE METRICS DASHBOARD	569
11.3. IDENTIFY METRICS FOR INDICATING OPTIMAL NODE RESOURCE USAGE	569
11.3.1. Top 3 containers with the most OOM kills in the last day	570
11.3.2. Failure rate for image pulls in the last hour	570
11.3.3. Nodes with system reserved memory utilization > 80%	571
11.3.4. Nodes with Kubelet system reserved memory utilization > 50%	571
11.3.5. Nodes with CRI-O system reserved memory utilization > 50%	571
11.3.6. Nodes with System Reserved CPU Utilization > 80%	572
11.3.7. Nodes with Kubelet system reserved CPU utilization > 50%	572
11.3.8. Nodes with CRI-O system reserved CPU utilization > 50%	573

11.4. CUSTOMIZING DASHBOARD QUERIES	573
CHAPTER 12. MANAGE SECURE SIGNATURES WITH SIGSTORE	574
12.1. ABOUT SIGSTORE	574
12.2. ABOUT CONFIGURING SIGSTORE SUPPORT	574
12.2.1. About cluster and image policy parameters	576
12.2.2. About modifying or removing image policies	578
12.3. CREATING A CLUSTER IMAGE POLICY CR	578
12.4. CREATING AN IMAGE POLICY CR	585

CHAPTER 1. OVERVIEW OF NODES

1.1. ABOUT NODES

A node is a virtual or bare-metal machine in a Kubernetes cluster. Worker nodes host your application containers, grouped as pods. The control plane nodes run services that are required to control the Kubernetes cluster. In OpenShift Container Platform, the control plane nodes contain more than just the Kubernetes services for managing the OpenShift Container Platform cluster.

Having stable and healthy nodes in a cluster is fundamental to the smooth functioning of your hosted application. In OpenShift Container Platform, you can access, manage, and monitor a node through the **Node** object representing the node. Using the OpenShift CLI (**oc**) or the web console, you can perform the following operations on a node.

The following components of a node are responsible for maintaining the running of pods and providing the Kubernetes runtime environment.

Container runtime

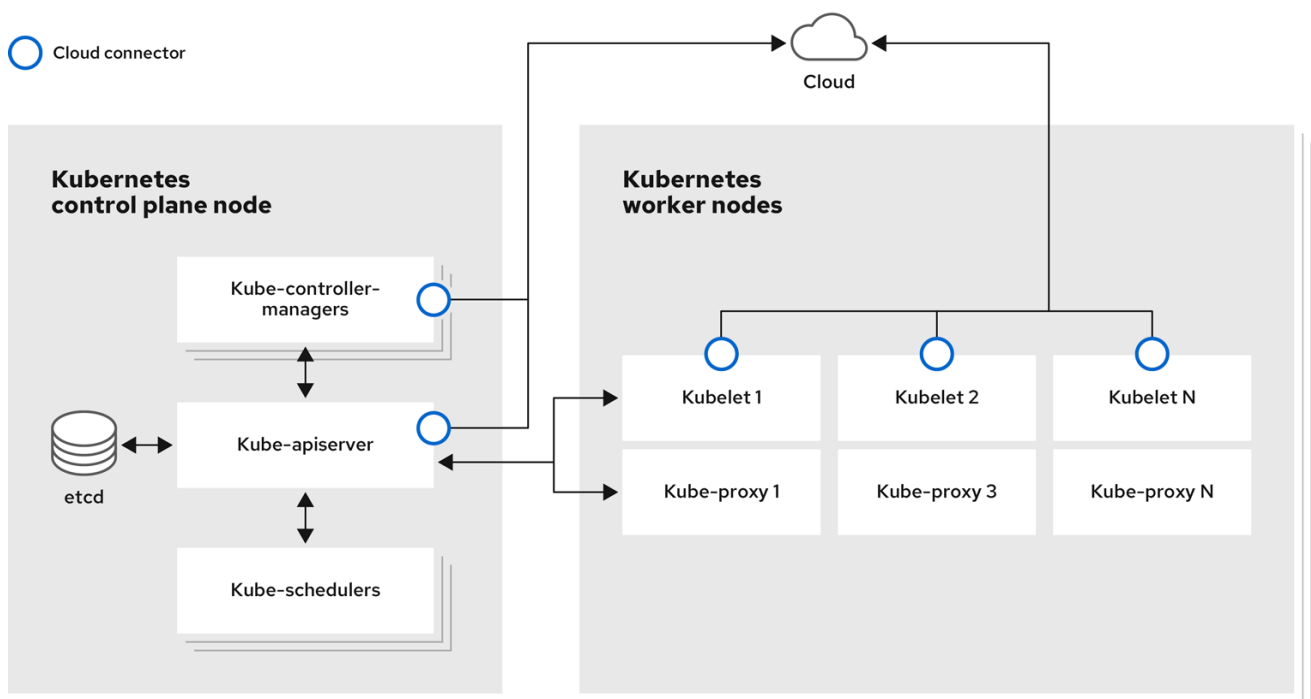
The container runtime is responsible for running containers. OpenShift Container Platform deploys the CRI-O container runtime on each of the Red Hat Enterprise Linux CoreOS (RHCOS) nodes in your cluster. The Windows Machine Config Operator (WMCO) deploys the containerd runtime on its Windows nodes.

Kubelet

Kubelet runs on nodes and reads the container manifests. It ensures that the defined containers have started and are running. The kubelet process maintains the state of work and the node server. Kubelet manages network rules and port forwarding. The kubelet manages containers that are created by Kubernetes only.

DNS

Cluster DNS is a DNS server which serves DNS records for Kubernetes services. Containers started by Kubernetes automatically include this DNS server in their DNS searches.



295_OpenShift_1222

1.1.1. Read operations

The read operations allow an administrator or a developer to get information about nodes in an OpenShift Container Platform cluster.

- [List all the nodes in a cluster](#) .
- Get information about a node, such as memory and CPU usage, health, status, and age.
- [List pods running on a node](#) .

1.1.2. Enhancement operations

OpenShift Container Platform allows you to do more than just access and manage nodes; as an administrator, you can perform the following tasks on nodes to make the cluster more efficient, application-friendly, and to provide a better environment for your developers.

- Manage node-level tuning for high-performance applications that require some level of kernel tuning by [using the Node Tuning Operator](#) .
- Enable TLS security profiles on the node to protect communication between the kubelet and the Kubernetes API server.
- [Run background tasks on nodes automatically with daemon sets](#) . You can create and use daemon sets to create shared storage, run a logging pod on every node, or deploy a monitoring agent on all nodes.
- [Free node resources using garbage collection](#) . You can ensure that your nodes are running efficiently by removing terminated containers and the images not referenced by any running pods.
- [Add kernel arguments to a set of nodes](#) .
- Configure an OpenShift Container Platform cluster to have worker nodes at the network edge (remote worker nodes). For information on the challenges of having remote worker nodes in an OpenShift Container Platform cluster and some recommended approaches for managing pods on a remote worker node, see [Using remote worker nodes at the network edge](#) .

1.2. ABOUT PODS

A pod is one or more containers deployed together on a node. As a cluster administrator, you can define a pod, assign it to run on a healthy node that is ready for scheduling, and manage. A pod runs as long as the containers are running. You cannot change a pod once it is defined and is running. Some operations you can perform when working with pods are:

1.2.1. Read operations

As an administrator, you can get information about pods in a project through the following tasks:

- [List pods associated with a project](#) , including information such as the number of replicas and restarts, current status, and age.
- [View pod usage statistics](#) such as CPU, memory, and storage consumption.

1.2.2. Management operations

The following list of tasks provides an overview of how an administrator can manage pods in an OpenShift Container Platform cluster.

- Control scheduling of pods using the advanced scheduling features available in OpenShift Container Platform:
 - Node-to-pod binding rules such as [pod affinity](#), [node affinity](#), and [anti-affinity](#).
 - [Node labels and selectors](#).
 - [Taints and tolerations](#).
 - [Pod topology spread constraints](#).
 - [Secondary scheduling](#).
- [Configure the descheduler to evict pods](#) based on specific strategies so that the scheduler reschedules the pods to more appropriate nodes.
- [Configure how pods behave after a restart using pod controllers and restart policies](#).
- [Limit both egress and ingress traffic on a pod](#).
- [Add and remove volumes to and from any object that has a pod template](#). A volume is a mounted file system available to all the containers in a pod. Container storage is ephemeral; you can use volumes to persist container data.

1.2.3. Enhancement operations

You can work with pods more easily and efficiently with the help of various tools and features available in OpenShift Container Platform. The following operations involve using those tools and features to better manage pods.

Operation	User	More information
Create and use a horizontal pod autoscaler.	Developer	You can use a horizontal pod autoscaler to specify the minimum and the maximum number of pods you want to run, as well as the CPU utilization or memory utilization your pods should target. Using a horizontal pod autoscaler, you can automatically scale pods .
Install and use a vertical pod autoscaler .	Administrator and developer	As an administrator, use a vertical pod autoscaler to better use cluster resources by monitoring the resources and the resource requirements of workloads. As a developer, use a vertical pod autoscaler to ensure your pods stay up during periods of high demand by scheduling pods to nodes that have enough resources for each pod.

Operation	User	More information
Provide access to external resources using device plugins.	Administrator	A device plugin is a gRPC service running on nodes (external to the kubelet), which manages specific hardware resources. You can deploy a device plugin to provide a consistent and portable solution to consume hardware devices across clusters.
Provide sensitive data to pods using the Secret object .	Administrator	Some applications need sensitive information, such as passwords and usernames. You can use the Secret object to provide such information to an application pod.

1.3. ABOUT CONTAINERS

A container is the basic unit of an OpenShift Container Platform application, which comprises the application code packaged along with its dependencies, libraries, and binaries. Containers provide consistency across environments and multiple deployment targets: physical servers, virtual machines (VMs), and private or public cloud.

Linux container technologies are lightweight mechanisms for isolating running processes and limiting access to only designated resources. As an administrator, You can perform various tasks on a Linux container, such as:

- [Copy files to and from a container](#) .
- [Allow containers to consume API objects](#) .
- [Execute remote commands in a container](#) .
- [Use port forwarding to access applications in a container](#) .

OpenShift Container Platform provides specialized containers called [Init containers](#). Init containers run before application containers and can contain utilities or setup scripts not present in an application image. You can use an Init container to perform tasks before the rest of a pod is deployed.

Apart from performing specific tasks on nodes, pods, and containers, you can work with the overall OpenShift Container Platform cluster to keep the cluster efficient and the application pods highly available.

1.4. ABOUT AUTOSCALING PODS ON A NODE

OpenShift Container Platform offers three tools that you can use to automatically scale the number of pods on your nodes and the resources allocated to pods.

Horizontal Pod Autoscaler

The Horizontal Pod Autoscaler (HPA) can automatically increase or decrease the scale of a replication controller or deployment configuration, based on metrics collected from the pods that belong to that replication controller or deployment configuration.

For more information, see [Automatically scaling pods with the horizontal pod autoscaler](#) .

Custom Metrics Autoscaler

The Custom Metrics Autoscaler can automatically increase or decrease the number of pods for a deployment, stateful set, custom resource, or job based on custom metrics that are not based only on CPU or memory.

For more information, see [Custom Metrics Autoscaler Operator overview](#) .

Vertical Pod Autoscaler

The Vertical Pod Autoscaler (VPA) can automatically review the historic and current CPU and memory resources for containers in pods and can update the resource limits and requests based on the usage values it learns.

For more information, see [Automatically adjust pod resource levels with the vertical pod autoscaler](#) .

1.5. GLOSSARY OF COMMON TERMS FOR OPENSIFT CONTAINER PLATFORM NODES

This glossary defines common terms that are used in the *node* content.

Container

It is a lightweight and executable image that comprises software and all its dependencies. Containers virtualize the operating system, as a result, you can run containers anywhere from a data center to a public or private cloud to even a developer's laptop.

Daemon set

Ensures that a replica of the pod runs on eligible nodes in an OpenShift Container Platform cluster.

egress

The process of data sharing externally through a network's outbound traffic from a pod.

garbage collection

The process of cleaning up cluster resources, such as terminated containers and images that are not referenced by any running pods.

Horizontal Pod Autoscaler(HPA)

Implemented as a Kubernetes API resource and a controller. You can use the HPA to specify the minimum and maximum number of pods that you want to run. You can also specify the CPU or memory utilization that your pods should target. The HPA scales out and scales in pods when a given CPU or memory threshold is crossed.

Ingress

Incoming traffic to a pod.

Job

A process that runs to completion. A job creates one or more pod objects and ensures that the specified pods are successfully completed.

Labels

You can use labels, which are key-value pairs, to organise and select subsets of objects, such as a pod.

Node

A worker machine in the OpenShift Container Platform cluster. A node can be either be a virtual machine (VM) or a physical machine.

Node Tuning Operator

You can use the Node Tuning Operator to manage node-level tuning by using the TuneD daemon. It ensures custom tuning specifications are 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.

Self Node Remediation Operator

The Operator runs on the cluster nodes and identifies and reboots nodes that are unhealthy.

Pod

One or more containers with shared resources, such as volume and IP addresses, running in your OpenShift Container Platform cluster. A pod is the smallest compute unit defined, deployed, and managed.

Toleration

Indicates that the pod is allowed (but not required) to be scheduled on nodes or node groups with matching taints. You can use tolerations to enable the scheduler to schedule pods with matching taints.

Taint

A core object that comprises a key, value, and effect. Taints and tolerations work together to ensure that pods are not scheduled on irrelevant nodes.

CHAPTER 2. WORKING WITH PODS

2.1. USING PODS

A *pod* is one or more containers deployed together on one host, and the smallest compute unit that can be defined, deployed, and managed.

2.1.1. Understanding pods

Pods are the rough equivalent of a machine instance (physical or virtual) to a Container. Each pod is allocated its own internal IP address, therefore owning its entire port space, and containers within pods can share their local storage and networking.

Pods have a lifecycle; they are defined, then they are assigned to run on a node, then they run until their container(s) exit or they are removed for some other reason. Pods, depending on policy and exit code, might be removed after exiting, or can be retained to enable access to the logs of their containers.

OpenShift Container Platform treats pods as largely immutable; changes cannot be made to a pod definition while it is running. OpenShift Container Platform implements changes by terminating an existing pod and recreating it with modified configuration, base image(s), or both. Pods are also treated as expendable, and do not maintain state when recreated. Therefore pods should usually be managed by higher-level controllers, rather than directly by users.



NOTE

For the maximum number of pods per OpenShift Container Platform node host, see the Cluster Limits.



WARNING

Bare pods that are not managed by a replication controller will be not rescheduled upon node disruption.

2.1.2. Example pod configurations

OpenShift Container Platform leverages the Kubernetes concept of a *pod*, which is one or more containers deployed together on one host, and the smallest compute unit that can be defined, deployed, and managed.

The following is an example definition of a pod. It demonstrates many features of pods, most of which are discussed in other topics and thus only briefly mentioned here:

Pod object definition (YAML)

```
kind: Pod
apiVersion: v1
metadata:
  name: example
  labels:
```



```

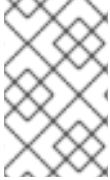
environment: production
app: abc ❶
spec:
  restartPolicy: Always ❷
  securityContext: ❸
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers: ❹
    - name: abc
      args:
        - sleep
        - "1000000"
      volumeMounts: ❺
        - name: cache-volume
          mountPath: /cache ❻
      image: registry.access.redhat.com/ubi7/ubi-init:latest ❼
      securityContext:
        allowPrivilegeEscalation: false
        runAsNonRoot: true
        capabilities:
          drop: ["ALL"]
      resources:
        limits:
          memory: "100Mi"
          cpu: "1"
        requests:
          memory: "100Mi"
          cpu: "1"
  volumes: ❽
    - name: cache-volume
      emptyDir:
        sizeLimit: 500Mi

```

- ❶ Pods can be "tagged" with one or more labels, which can then be used to select and manage groups of pods in a single operation. The labels are stored in key/value format in the **metadata** hash.
- ❷ The pod restart policy with possible values **Always**, **OnFailure**, and **Never**. The default value is **Always**.
- ❸ OpenShift Container Platform defines a security context for containers which specifies whether they are allowed to run as privileged containers, run as a user of their choice, and more. The default context is very restrictive but administrators can modify this as needed.
- ❹ **containers** specifies an array of one or more container definitions.
- ❺ The container specifies where external storage volumes are mounted within the container.
- ❻ Specify the volumes to provide for the pod. Volumes mount at the specified path. Do not mount to the container root, `/`, or any path that is the same in the host and the container. This can corrupt your host system if the container is sufficiently privileged, such as the host `/dev/pts` files. It is safe to mount the host by using `/host`.
- ❼ Each container in the pod is instantiated from its own container image.

- 8 The pod defines storage volumes that are available to its container(s) to use.

If you attach persistent volumes that have high file counts to pods, those pods can fail or can take a long time to start. For more information, see [When using Persistent Volumes with high file counts in OpenShift, why do pods fail to start or take an excessive amount of time to achieve "Ready" state?](#).



NOTE

This pod definition does not include attributes that are filled by OpenShift Container Platform automatically after the pod is created and its lifecycle begins. The [Kubernetes pod documentation](#) has details about the functionality and purpose of pods.

2.1.3. Understanding resource requests and limits

You can specify CPU and memory requests and limits for pods by using a pod spec, as shown in "Example pod configurations", or the specification for the controlling object of the pod.

CPU and memory *requests* specify the minimum amount of a resource that a pod needs to run, helping OpenShift Container Platform to schedule pods on nodes with sufficient resources.

CPU and memory *limits* define the maximum amount of a resource that a pod can consume, preventing the pod from consuming excessive resources and potentially impacting other pods on the same node.

CPU and memory requests and limits are processed by using the following principles:

- CPU limits are enforced by using CPU throttling. When a container approaches its CPU limit, the kernel restricts access to the CPU specified as the container's limit. As such, a CPU limit is a hard limit that the kernel enforces. OpenShift Container Platform can allow a container to exceed its CPU limit for extended periods of time. However, container runtimes do not terminate pods or containers for excessive CPU usage.
CPU limits and requests are measured in CPU units. One CPU unit is equivalent to 1 physical CPU core or 1 virtual core, depending on whether the node is a physical host or a virtual machine running inside a physical machine. Fractional requests are allowed. For example, when you define a container with a CPU request of **0.5**, you are requesting half as much CPU time than if you asked for **1.0** CPU. For CPU units, **0.1** is equivalent to the **100m**, which can be read as *one hundred millicpu* or *one hundred millicores*. A CPU resource is always an absolute amount of resource, and is never a relative amount.



NOTE

By default, the smallest amount of CPU that can be allocated to a pod is 10 mCPU. You can request resource limits lower than 10 mCPU in a pod spec. However, the pod would still be allocated 10 mCPU.

- Memory limits are enforced by the kernel by using out of memory (OOM) kills. When a container uses more than its memory limit, the kernel can terminate that container. However, terminations happen only when the kernel detects memory pressure. As such, a container that over allocates memory might not be immediately killed. This means memory limits are enforced reactively. A container can use more memory than its memory limit. If it does, the container can get killed. You can express memory as a plain integer or as a fixed-point number by using one of these quantity suffixes: **E**, **P**, **T**, **G**, **M**, or **k**. You can also use the power-of-two equivalents: **Ei**, **Pi**, **Ti**, **Gi**, **Mi**, or **Ki**.

If the node where a pod is running has enough of a resource available, it is possible for a container to use more CPU or memory resources than it requested. However, the container cannot exceed the corresponding limit. For example, if you set a container memory request of **256 MiB**, and that container is in a pod scheduled to a node with **8GiB** of memory and no other pods, the container can try to use more memory than the requested **256 MiB**.

This behavior does not apply to CPU and memory limits. These limits are applied by the kubelet and the container runtime, and are enforced by the kernel. On Linux nodes, the kernel enforces limits by using cgroups.

For Linux workloads, you can specify huge page resources. Huge pages are a Linux-specific feature where the node kernel allocates blocks of memory that are much larger than the default page size. For example, on a system where the default page size is 4KiB, you could specify a higher limit. For more information on huge pages, see "Huge pages".

2.1.4. Additional resources

- For more information on pods and storage see [Understanding persistent storage](#) and [Understanding ephemeral storage](#).
- [Example pod configurations](#)
- [Huge pages](#)

2.2. VIEWING PODS

As an administrator, you can view cluster pods, check their health, and evaluate the overall health of the cluster. You can also view a list of pods associated with a specific project or view usage statistics about pods. Regularly viewing pods can help you detect problems early, track resource usage, and ensure cluster stability.

2.2.1. Viewing pods in a project

You can display pod usage statistics, such as CPU, memory, and storage consumption, to monitor container runtime environments and ensure efficient resource use.

Procedure

1. Change to the project by entering the following command:

```
$ oc project <project_name>
```

2. Obtain a list of pods by entering the following command:

```
$ oc get pods
```

Example output

```
NAME                READY STATUS  RESTARTS  AGE
console-698d866b78-bnshf 1/1   Running  2         165m
console-698d866b78-m87pm 1/1   Running  2         165m
```

- Optional: Add the **-o wide** flags to view the pod IP address and the node where the pod is located. For example:

```
$ oc get pods -o wide
```

Example output

```
NAME                                READY STATUS  RESTARTS  AGE  IP              NODE
NOMINATED NODE
console-698d866b78-bnshf 1/1   Running  2         166m  10.128.0.24    ip-10-0-152-71.ec2.internal <none>
console-698d866b78-m87pm 1/1   Running  2         166m  10.129.0.23    ip-10-0-173-237.ec2.internal <none>
```

2.2.2. Viewing pod usage statistics

You can display usage statistics about pods, which provide the runtime environments for containers. These usage statistics include CPU, memory, and storage consumption.

Prerequisites

- You must have **cluster-reader** permission to view the usage statistics.
- Metrics must be installed to view the usage statistics.

Procedure

- View the usage statistics by entering the following command:

```
$ oc adm top pods -n <namespace>
```

Example output

```
NAME                                CPU(cores)  MEMORY(bytes)
console-7f58c69899-q8c8k  0m          22Mi
console-7f58c69899-xhbgg  0m          25Mi
downloads-594fccf94-bcxk8  3m          18Mi
downloads-594fccf94-kv4p6  2m          15Mi
```

- Optional: Add the **--selector="** label to view usage statistics for pods with labels. Note that you must choose the label query to filter on, such as **=**, **==**, or **!=**. For example:

```
$ oc adm top pod --selector='<pod_name>'
```

2.2.3. Viewing resource logs

You can view logs for resources in the OpenShift CLI (oc) or web console. Logs display from the end (or tail) by default. Viewing logs for resources can help you troubleshoot issues and monitor resource behavior.

2.2.3.1. Viewing resource logs by using the web console

Use the following procedure to view resource logs by using the OpenShift Container Platform web console.

Procedure

1. In the OpenShift Container Platform console, navigate to **Workloads → Pods** or navigate to the pod through the resource you want to investigate.



NOTE

Some resources, such as builds, do not have pods to query directly. In such instances, you can locate the **Logs** link on the **Details** page for the resource.

2. Select a project from the drop-down menu.
3. Click the name of the pod you want to investigate.
4. Click **Logs**.

2.2.3.2. Viewing resource logs by using the CLI

Use the following procedure to view resource logs by using the command-line interface (CLI).

Prerequisites

- Access to the OpenShift CLI (**oc**).

Procedure

- View the log for a specific pod by entering the following command:

```
$ oc logs -f <pod_name> -c <container_name>
```

where:

-f

Optional: Specifies that the output follows what is being written into the logs.

<pod_name>

Specifies the name of the pod.

<container_name>

Optional: Specifies the name of a container. When a pod has more than one container, you must specify the container name.

For example:

```
$ oc logs -f ruby-57f7f4855b-znl92 -c ruby
```

- View the log for a specific resource by entering the following command:

```
$ oc logs <object_type>/<resource_name>
```

For example:

```
$ oc logs deployment/ruby
```

2.3. CONFIGURING AN OPENSIFT CONTAINER PLATFORM CLUSTER FOR PODS

As an administrator, you can create and maintain an efficient cluster for pods.

By keeping your cluster efficient, you can provide a better environment for your developers using such tools as what a pod does when it exits, ensuring that the required number of pods is always running, when to restart pods designed to run only once, limit the bandwidth available to pods, and how to keep pods running during disruptions.

2.3.1. Configuring how pods behave after restart

A pod restart policy determines how OpenShift Container Platform responds when Containers in that pod exit. The policy applies to all Containers in that pod.

The possible values are:

- **Always** - Tries restarting a successfully exited Container on the pod continuously, with an exponential back-off delay (10s, 20s, 40s) capped at 5 minutes. The default is **Always**.
- **OnFailure** - Tries restarting a failed Container on the pod with an exponential back-off delay (10s, 20s, 40s) capped at 5 minutes.
- **Never** - Does not try to restart exited or failed Containers on the pod. Pods immediately fail and exit.

After the pod is bound to a node, the pod will never be bound to another node. This means that a controller is necessary in order for a pod to survive node failure:

Condition	Controller Type	Restart Policy
Pods that are expected to terminate (such as batch computations)	Job	OnFailure or Never
Pods that are expected to not terminate (such as web servers)	Replication controller	Always .
Pods that must run one-per-machine	Daemon set	Any

If a Container on a pod fails and the restart policy is set to **OnFailure**, the pod stays on the node and the Container is restarted. If you do not want the Container to restart, use a restart policy of **Never**.

If an entire pod fails, OpenShift Container Platform starts a new pod. Developers must address the possibility that applications might be restarted in a new pod. In particular, applications must handle temporary files, locks, incomplete output, and so forth caused by previous runs.



NOTE

Kubernetes architecture expects reliable endpoints from cloud providers. When a cloud provider is down, the kubelet prevents OpenShift Container Platform from restarting.

If the underlying cloud provider endpoints are not reliable, do not install a cluster using cloud provider integration. Install the cluster as if it was in a no-cloud environment. It is not recommended to toggle cloud provider integration on or off in an installed cluster.

For details on how OpenShift Container Platform uses restart policy with failed Containers, see the [Example States](#) in the Kubernetes documentation.

2.3.2. Limiting the bandwidth available to pods

You can apply quality-of-service traffic shaping to a pod and effectively limit its available bandwidth. Egress traffic (from the pod) is handled by policing, which simply drops packets in excess of the configured rate. Ingress traffic (to the pod) is handled by shaping queued packets to effectively handle data. The limits you place on a pod do not affect the bandwidth of other pods.

Procedure

To limit the bandwidth on a pod:

1. Write an object definition JSON file, and specify the data traffic speed using **kubernetes.io/ingress-bandwidth** and **kubernetes.io/egress-bandwidth** annotations. For example, to limit both pod egress and ingress bandwidth to 10M/s:

Limited Pod object definition

```
{
  "kind": "Pod",
  "spec": {
    "containers": [
      {
        "image": "openshift/hello-openshift",
        "name": "hello-openshift"
      }
    ]
  },
  "apiVersion": "v1",
  "metadata": {
    "name": "iperf-slow",
    "annotations": {
      "kubernetes.io/ingress-bandwidth": "10M",
      "kubernetes.io/egress-bandwidth": "10M"
    }
  }
}
```

2. Create the pod using the object definition:

```
$ oc create -f <file_or_dir_path>
```

2.3.3. 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 121m	openshift-apiserver-pdb	N/A	1
openshift-cloud-controller-manager 125m	aws-cloud-controller-manager	1	N/A
openshift-cloud-credential-operator 117m	pod-identity-webhook	1	N/A
openshift-cluster-csi-drivers 121m	aws-ebs-csi-driver-controller-pdb	N/A	1
openshift-cluster-storage-operator 122m	csi-snapshot-controller-pdb	N/A	1
openshift-cluster-storage-operator 122m	csi-snapshot-webhook-pdb	N/A	1
openshift-console 116m	console	N/A	1
#...			

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.

2.3.3.1. 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**

Or:

```
apiVersion: policy/v1 1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  maxUnavailable: 25% 2
  selector: 3
    matchLabels:
      name: my-pod
```

- 1** **PodDisruptionBudget** is part of the **policy/v1** API group.
- 2** The maximum number of pods that can be unavailable simultaneously. This can be either an integer or a string specifying a percentage, for example, **20%**.
- 3** A label query over a set of resources. The result of **matchLabels** and **matchExpressions** are logically conjoined. Leave this parameter blank, for example **selector {}**, to select all pods in the project.

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

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

2.3.3.2. 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.

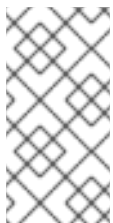
You can choose one of the following policies:

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 \(Kubernetes documentation\)](#)

2.3.4. Preventing pod removal using critical pods

There are a number of core components that are critical to a fully functional cluster, but, run on a regular cluster node rather than the master. A cluster might stop working properly if a critical add-on is evicted.

Pods marked as critical are not allowed to be evicted.

Procedure

To make a pod critical:

1. Create a **Pod** spec or edit existing pods to include the **system-cluster-critical** priority class:

```

apiVersion: v1
kind: Pod
metadata:
  name: my-pdb
spec:
  template:
    metadata:
      name: critical-pod
    priorityClassName: system-cluster-critical 1
# ...

```

- 1 Default priority class for pods that should never be evicted from a node.

Alternatively, you can specify **system-node-critical** for pods that are important to the cluster but can be removed if necessary.

2. Create the pod:

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

2.3.5. Reducing pod timeouts when using persistent volumes with high file counts

If a storage volume contains many files (~1,000,000 or greater), you might experience pod timeouts.

This can occur because, when volumes are mounted, OpenShift Container Platform recursively changes the ownership and permissions of the contents of each volume in order to match the **fsGroup** specified in a pod's **securityContext**. For large volumes, checking and changing the ownership and permissions can be time consuming, resulting in a very slow pod startup.

You can reduce this delay by applying one of the following workarounds:

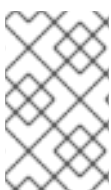
- Use a security context constraint (SCC) to skip the SELinux relabeling for a volume.
- Use the **fsGroupChangePolicy** field inside an SCC to control the way that OpenShift Container Platform checks and manages ownership and permissions for a volume.
- Use the Cluster Resource Override Operator to automatically apply an SCC to skip the SELinux relabeling.
- Use a runtime class to skip the SELinux relabeling for a volume.

For information, see [When using Persistent Volumes with high file counts in OpenShift, why do pods fail to start or take an excessive amount of time to achieve "Ready" state?](#).

2.4. AUTOMATICALLY SCALING PODS WITH THE HORIZONTAL POD AUTOSCALER

As a developer, you can use a horizontal pod autoscaler (HPA) to specify how OpenShift Container Platform should automatically increase or decrease the scale of a replication controller or deployment configuration, based on metrics collected from the pods that belong to that replication controller or deployment configuration. You can create an HPA for any deployment, deployment config, replica set, replication controller, or stateful set.

For information on scaling pods based on custom metrics, see [Automatically scaling pods based on custom metrics](#).



NOTE

It is recommended to use a **Deployment** object or **ReplicaSet** object unless you need a specific feature or behavior provided by other objects. For more information on these objects, see [Understanding deployments](#).

2.4.1. Understanding horizontal pod autoscalers

You can create a horizontal pod autoscaler to specify the minimum and maximum number of pods you want to run, and the CPU usage or memory usage your pods should target.

After you create a horizontal pod autoscaler, OpenShift Container Platform begins to query the CPU, memory, or both resource metrics on the pods. When these metrics are available, the horizontal pod autoscaler computes the ratio of the current metric use with the intended metric use, and scales up or down as needed. The query and scaling occurs at a regular interval, but can take one to two minutes before metrics become available.

For replication controllers, this scaling corresponds directly to the replicas of the replication controller. For deployment, scaling corresponds directly to the replica count of the deployment. Note that autoscaling applies only to the latest deployment in the **Complete** phase.

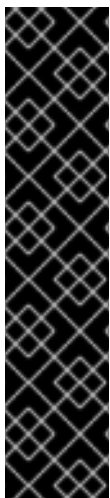
OpenShift Container Platform automatically accounts for resources and prevents unnecessary autoscaling during resource spikes, such as during start up. Pods in the **unready** state have **0 CPU** usage when scaling up and the autoscaler ignores the pods when scaling down. Pods without known metrics have **0% CPU** usage when scaling up and **100% CPU** when scaling down. This allows for more stability during the HPA decision. To use this feature, you must configure readiness checks to determine if a new pod is ready for use.

To use horizontal pod autoscalers, your cluster administrator must have properly configured cluster metrics.

The following metrics are supported by horizontal pod autoscalers:

Table 2.1. Supported metrics

Metric	Description	API version
CPU utilization	Number of CPU cores used. You can use this to calculate a percentage of the pod's requested CPU.	autoscaling/v1, autoscaling/v2
Memory utilization	Amount of memory used. You can use this to calculate a percentage of the pod's requested memory.	autoscaling/v2



IMPORTANT

For memory-based autoscaling, memory usage must increase and decrease proportionally to the replica count. On average:

- An increase in replica count must lead to an overall decrease in memory (working set) usage per-pod.
- A decrease in replica count must lead to an overall increase in per-pod memory usage.

Use the OpenShift Container Platform web console to check the memory behavior of your application and ensure that your application meets these requirements before using memory-based autoscaling.

The following example shows autoscaling for the **hello-node Deployment** object. The initial deployment requires 3 pods. The HPA object increases the minimum to 5. If CPU usage on the pods reaches 75%, the pods increase to 7:

```
$ oc autoscale deployment/hello-node --min=5 --max=7 --cpu-percent=75
```

Example output

```
horizontalpodautoscaler.autoscaling/hello-node autoscaled
```

Sample YAML to create an HPA for the `hello-node` deployment object with `minReplicas` set to 3

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: hello-node
  namespace: default
spec:
  maxReplicas: 7
  minReplicas: 3
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: hello-node
  targetCPUUtilizationPercentage: 75
status:
  currentReplicas: 5
  desiredReplicas: 0
```

After you create the HPA, you can view the new state of the deployment by running the following command:

```
$ oc get deployment hello-node
```

There are now 5 pods in the deployment:

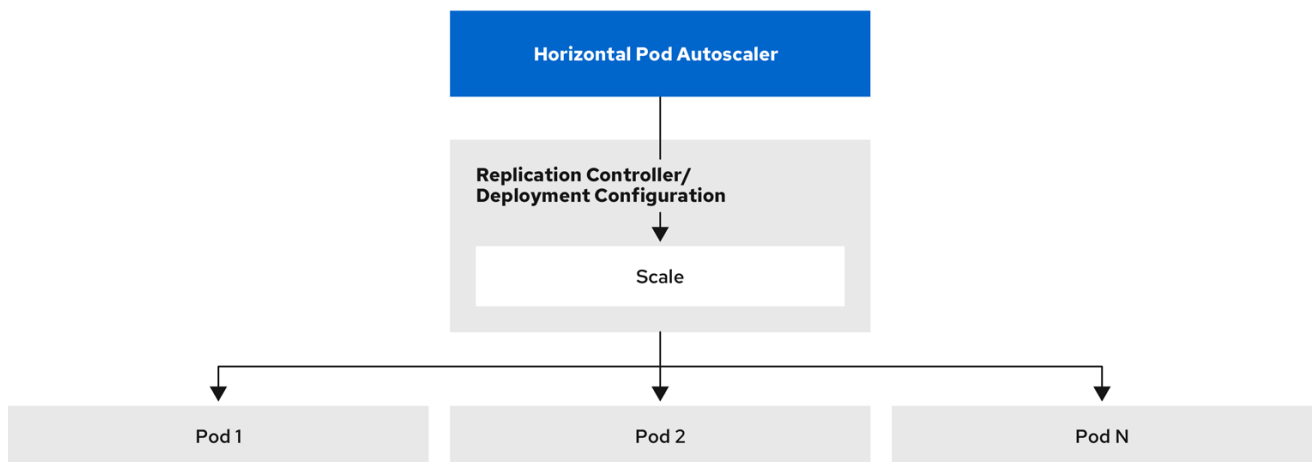
Example output

NAME	REVISION	DESIRED	CURRENT	TRIGGERED BY
hello-node	1	5	5	config

2.4.2. How does the HPA work?

The horizontal pod autoscaler (HPA) extends the concept of pod auto-scaling. The HPA lets you create and manage a group of load-balanced nodes. The HPA automatically increases or decreases the number of pods when a given CPU or memory threshold is crossed.

Figure 2.1. High level workflow of the HPA



223_OpenShift_0222

The HPA is an API resource in the Kubernetes autoscaling API group. The autoscaler works as a control loop with a default of 15 seconds for the sync period. During this period, the controller manager queries the CPU, memory utilization, or both, against what is defined in the YAML file for the HPA. The controller manager obtains the utilization metrics from the resource metrics API for per-pod resource metrics like CPU or memory, for each pod that is targeted by the HPA.

If a utilization value target is set, the controller calculates the utilization value as a percentage of the equivalent resource request on the containers in each pod. The controller then takes the average of utilization across all targeted pods and produces a ratio that is used to scale the number of desired replicas. The HPA is configured to fetch metrics from **metrics.k8s.io**, which is provided by the metrics server. Because of the dynamic nature of metrics evaluation, the number of replicas can fluctuate during scaling for a group of replicas.

**NOTE**

To implement the HPA, all targeted pods must have a resource request set on their containers.

2.4.3. About requests and limits

The scheduler uses the resource request that you specify for containers in a pod, to decide which node to place the pod on. The kubelet enforces the resource limit that you specify for a container to ensure that the container is not allowed to use more than the specified limit. The kubelet also reserves the request amount of that system resource specifically for that container to use.

How to use resource metrics?

In the pod specifications, you must specify the resource requests, such as CPU and memory. The HPA uses this specification to determine the resource utilization and then scales the target up or down.

For example, the HPA object uses the following metric source:

```

type: Resource
resource:
  name: cpu
  target:
    type: Utilization
    averageUtilization: 60
  
```

-

In this example, the HPA keeps the average utilization of the pods in the scaling target at 60%. Utilization is the ratio between the current resource usage to the requested resource of the pod.

2.4.4. Best practices

For optimal performance, configure resource requests for all pods. To prevent frequent replica fluctuations, configure the cooldown period.

All pods must have resource requests configured

The HPA makes a scaling decision based on the observed CPU or memory usage values of pods in an OpenShift Container Platform cluster. Utilization values are calculated as a percentage of the resource requests of each pod. Missing resource request values can affect the optimal performance of the HPA.

For more information, see "Understanding resource requests and limits".

Configure the cool down period

During horizontal pod autoscaling, there might be a rapid scaling of events without a time gap. Configure the cool down period to prevent frequent replica fluctuations. You can specify a cool down period by configuring the **stabilizationWindowSeconds** field. The stabilization window is used to restrict the fluctuation of replicas count when the metrics used for scaling keep fluctuating. The autoscaling algorithm uses this window to infer a previous required state and avoid unwanted changes to workload scale.

For example, a stabilization window is specified for the **scaleDown** field:

```
behavior:
  scaleDown:
    stabilizationWindowSeconds: 300
```

In the previous example, all intended states for the past 5 minutes are considered. This approximates a rolling maximum, and avoids having the scaling algorithm often remove pods only to trigger recreating an equal pod just moments later.

For more information, see "Scaling policies".

Additional resources

- [Understanding resource requests and limits](#)
- [Scaling policies](#)

2.4.4.1. Scaling policies

Use the **autoscaling/v2** API to add *scaling policies* to a horizontal pod autoscaler. A scaling policy controls how the OpenShift Container Platform horizontal pod autoscaler (HPA) scales pods. Use scaling policies to restrict the rate that HPAs scale pods up or down by setting a specific number or specific percentage to scale in a specified period of time. You can also define a *stabilization window*, which uses previously computed required states to control scaling if the metrics are fluctuating. You can create multiple policies for the same scaling direction, and determine the policy to use, based on the amount of change. You can also restrict the scaling by timed iterations. The HPA scales pods during an iteration, then performs scaling, as needed, in further iterations.

Sample HPA object with a scaling policy

```

apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-resource-metrics-memory
  namespace: default
spec:
  behavior:
    scaleDown: 1
    policies: 2
    - type: Pods 3
      value: 4 4
      periodSeconds: 60 5
    - type: Percent
      value: 10 6
      periodSeconds: 60
    selectPolicy: Min 7
    stabilizationWindowSeconds: 300 8
  scaleUp: 9
    policies:
    - type: Pods
      value: 5 10
      periodSeconds: 70
    - type: Percent
      value: 12 11
      periodSeconds: 80
    selectPolicy: Max
    stabilizationWindowSeconds: 0
  ...

```

- 1 Specifies the direction for the scaling policy, either **scaleDown** or **scaleUp**. This example creates a policy for scaling down.
- 2 Defines the scaling policy.
- 3 Determines if the policy scales by a specific number of pods or a percentage of pods during each iteration. The default value is **pods**.
- 4 Limits the amount of scaling, either the number of pods or percentage of pods, during each iteration. There is no default value for scaling down by number of pods.
- 5 Determines the length of a scaling iteration. The default value is **15** seconds.
- 6 The default value for scaling down by percentage is 100%.
- 7 Determines the policy to use first, if multiple policies are defined. Specify **Max** to use the policy that allows the highest amount of change, **Min** to use the policy that allows the lowest amount of change, or **Disabled** to prevent the HPA from scaling in that policy direction. The default value is **Max**.
- 8 Determines the time period the HPA reviews the required states. The default value is **0**.
- 9 This example creates a policy for scaling up.

- 10 Limits the amount of scaling up by the number of pods. The default value for scaling up the number of pods is 4%.
- 11 Limits the amount of scaling up by the percentage of pods. The default value for scaling up by percentage is 100%.

Example policy for scaling down

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-resource-metrics-memory
  namespace: default
spec:
  ...
  minReplicas: 20
  ...
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
        - type: Pods
          value: 4
          periodSeconds: 30
        - type: Percent
          value: 10
          periodSeconds: 60
      selectPolicy: Max
    scaleUp:
      selectPolicy: Disabled
```

In this example, when the number of pods is greater than 40, the percent-based policy is used for scaling down, as that policy results in a larger change, as required by the **selectPolicy**.

If there are 80 pod replicas, in the first iteration the HPA reduces the pods by 8, which is 10% of the 80 pods (based on the **type: Percent** and **value: 10** parameters), over one minute (**periodSeconds: 60**). For the next iteration, the number of pods is 72. The HPA calculates that 10% of the remaining pods is 7.2, which it rounds up to 8 and scales down 8 pods. On each subsequent iteration, the number of pods to be scaled is re-calculated based on the number of remaining pods. When the number of pods falls to less than 40, the pods-based policy is applied, because the pod-based number is greater than the percent-based number. The HPA reduces 4 pods at a time (**type: Pods** and **value: 4**), over 30 seconds (**periodSeconds: 30**), until there are 20 replicas remaining (**minReplicas**).

The **selectPolicy: Disabled** parameter prevents the HPA from scaling up the pods. You can manually scale up by adjusting the number of replicas in the replica set or deployment set, if needed.

If set, you can view the scaling policy by using the **oc edit** command:

```
$ oc edit hpa hpa-resource-metrics-memory
```

Example output

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
```

```

metadata:
  annotations:
    autoscaling.alpha.kubernetes.io/behavior:\
'{"ScaleUp":{"StabilizationWindowSeconds":0,"SelectPolicy":"Max","Policies":\
[{"Type":"Pods","Value":4,"PeriodSeconds":15},{ "Type":"Percent","Value":100,"PeriodSeconds":15}]\
"ScaleDown":{"StabilizationWindowSeconds":300,"SelectPolicy":"Min","Policies":\
[{"Type":"Pods","Value":4,"PeriodSeconds":60},{ "Type":"Percent","Value":10,"PeriodSeconds":60}]}}'
...

```

2.4.5. Creating a horizontal pod autoscaler by using the web console

From the web console, you can create a horizontal pod autoscaler (HPA) that specifies the minimum and maximum number of pods you want to run on a **Deployment** or **DeploymentConfig** object. You can also define the amount of CPU or memory usage that your pods should target.



NOTE

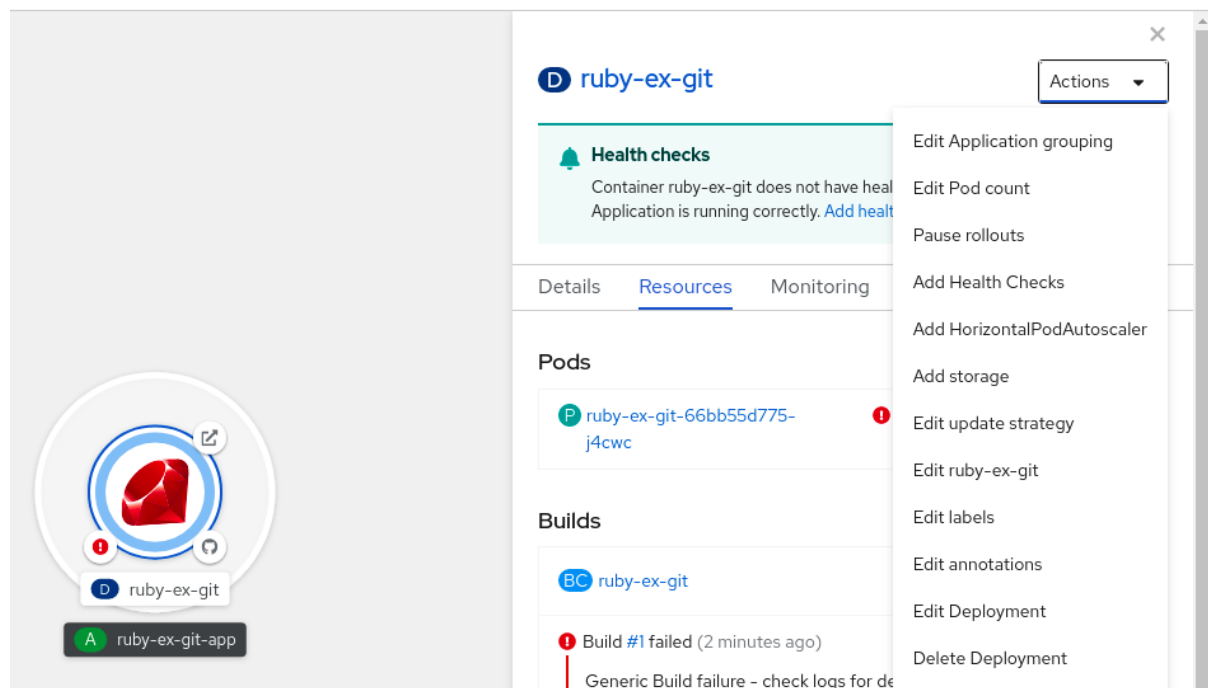
An HPA cannot be added to deployments that are part of an Operator-backed service, Knative service, or Helm chart.

Procedure

To create an HPA in the web console:

1. In the **Topology** view, click the node to reveal the side pane.
2. From the **Actions** drop-down list, select **Add HorizontalPodAutoscaler** to open the **Add HorizontalPodAutoscaler** form.

Figure 2.2. Add HorizontalPodAutoscaler



3. From the **Add HorizontalPodAutoscaler** form, define the name, minimum and maximum pod limits, the CPU and memory usage, and click **Save**.

**NOTE**

If any of the values for CPU and memory usage are missing, a warning is displayed.

2.4.5.1. Editing a horizontal pod autoscaler by using the web console

From the web console, you can modify a horizontal pod autoscaler (HPA) that specifies the minimum and maximum number of pods you want to run on a **Deployment** or **DeploymentConfig** object. You can also define the amount of CPU or memory usage that your pods should target.

Procedure

1. In the **Topology** view, click the node to reveal the side pane.
2. From the **Actions** drop-down list, select **Edit HorizontalPodAutoscaler** to open the **Edit Horizontal Pod Autoscaler** form.
3. From the **Edit Horizontal Pod Autoscaler** form, edit the minimum and maximum pod limits and the CPU and memory usage, and click **Save**.

**NOTE**

While creating or editing the horizontal pod autoscaler in the web console, you can switch from **Form view** to **YAML view**.

2.4.5.2. Removing a horizontal pod autoscaler by using the web console

You can remove a horizontal pod autoscaler (HPA) in the web console.

Procedure

1. In the **Topology** view, click the node to reveal the side panel.
2. From the **Actions** drop-down list, select **Remove HorizontalPodAutoscaler**.
3. In the confirmation window, click **Remove** to remove the HPA.

2.4.6. Creating a horizontal pod autoscaler by using the CLI

Using the OpenShift Container Platform CLI, you can create a horizontal pod autoscaler (HPA) to automatically scale an existing **Deployment**, **DeploymentConfig**, **ReplicaSet**, **ReplicationController**, or **StatefulSet** object. The HPA scales the pods associated with that object to maintain the CPU or memory resources that you specify.

You can autoscale based on CPU or memory use by specifying a percentage of resource usage or a specific value, as described in the following sections.

The HPA increases and decreases the number of replicas between the minimum and maximum numbers to maintain the specified resource use across all pods.

2.4.6.1. Creating a horizontal pod autoscaler for a percent of CPU use

Using the OpenShift Container Platform CLI, you can create a horizontal pod autoscaler (HPA) to automatically scale an existing object based on percent of CPU use. The HPA scales the pods associated with that object to maintain the CPU use that you specify.

When autoscaling for a percent of CPU use, you can use the **oc autoscale** command to specify the minimum and maximum number of pods that you want to run at any given time and the average CPU use your pods should target. If you do not specify a minimum, the pods are given default values from the OpenShift Container Platform server.



NOTE

Use a **Deployment** object or **ReplicaSet** object unless you need a specific feature or behavior provided by other objects.

Prerequisites

To use horizontal pod autoscalers, your cluster administrator must have properly configured cluster metrics. You can use the **oc describe PodMetrics <pod-name>** command to determine if metrics are configured. If metrics are configured, the output appears similar to the following, with **Cpu** and **Memory** displayed under **Usage**.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

Example output

```
Name:      openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace: openshift-kube-scheduler
Labels:    <none>
Annotations: <none>
API Version: metrics.k8s.io/v1beta1
Containers:
  Name: wait-for-host-port
  Usage:
    Memory: 0
  Name: scheduler
  Usage:
    Cpu: 8m
    Memory: 45440Ki
Kind:      PodMetrics
Metadata:
  Creation Timestamp: 2019-05-23T18:47:56Z
  Self Link:         /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-scheduler/pods/openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
  Timestamp:         2019-05-23T18:47:56Z
  Window:            1m0s
  Events:            <none>
```

Procedure

1. Create a **HorizontalPodAutoscaler** object for an existing object:

```
$ oc autoscale <object_type>/<name> \1
--min <number> \2
```

```
--max <number> \ 3
--cpu-percent=<percent> 4
```

- 1 Specify the type and name of the object to autoscale. The object must exist and be a **Deployment**, **DeploymentConfig/dc**, **ReplicaSet/rs**, **ReplicationController/rc**, or **StatefulSet**.
- 2 Optional: Specify the minimum number of replicas when scaling down.
- 3 Specify the maximum number of replicas when scaling up.
- 4 Specify the target average CPU use over all the pods, represented as a percent of requested CPU. If not specified or negative, a default autoscaling policy is used.

For example, the following command shows autoscaling for the **hello-node** deployment object. The initial deployment requires 3 pods. The HPA object increases the minimum to 5. If CPU usage on the pods reaches 75%, the pods will increase to 7:

```
$ oc autoscale deployment/hello-node --min=5 --max=7 --cpu-percent=75
```

2. Create the horizontal pod autoscaler:

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

Verification

- Ensure that the horizontal pod autoscaler was created:

```
$ oc get hpa cpu-autoscale
```

Example output

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS
AGE					
cpu-autoscale	Deployment/example	173m/500m	1	10	1
					20m

2.4.6.2. Creating a horizontal pod autoscaler for a specific CPU value

Using the OpenShift Container Platform CLI, you can create a horizontal pod autoscaler (HPA) to automatically scale an existing object based on a specific CPU value by creating a **HorizontalPodAutoscaler** object with the target CPU and pod limits. The HPA scales the pods associated with that object to maintain the CPU use that you specify.



NOTE

Use a **Deployment** object or **ReplicaSet** object unless you need a specific feature or behavior provided by other objects.

Prerequisites

To use horizontal pod autoscalers, your cluster administrator must have properly configured cluster metrics. You can use the **oc describe PodMetrics <pod-name>** command to determine if metrics are

configured. If metrics are configured, the output appears similar to the following, with **Cpu** and **Memory** displayed under **Usage**.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

Example output

```
Name:      openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace: openshift-kube-scheduler
Labels:    <none>
Annotations: <none>
API Version: metrics.k8s.io/v1beta1
Containers:
  Name: wait-for-host-port
  Usage:
    Memory: 0
  Name: scheduler
  Usage:
    Cpu: 8m
    Memory: 45440Ki
Kind: PodMetrics
Metadata:
  Creation Timestamp: 2019-05-23T18:47:56Z
  Self Link: /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-scheduler/pods/openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
  Timestamp: 2019-05-23T18:47:56Z
  Window: 1m0s
  Events: <none>
```

Procedure

1. Create a YAML file similar to the following for an existing object:

```
apiVersion: autoscaling/v2 1
kind: HorizontalPodAutoscaler
metadata:
  name: cpu-autoscale 2
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1 3
    kind: Deployment 4
    name: example 5
  minReplicas: 1 6
  maxReplicas: 10 7
  metrics: 8
  - type: Resource
    resource:
      name: cpu 9
      target:
        type: AverageValue 10
        averageValue: 500m 11
```

- 1 Use the **autoscaling/v2** API.
- 2 Specify a name for this horizontal pod autoscaler object.
- 3 Specify the API version of the object to scale:
 - For a **Deployment**, **ReplicaSet**, **StatefulSet** object, use **apps/v1**.
 - For a **ReplicationController**, use **v1**.
 - For a **DeploymentConfig**, use **apps.openshift.io/v1**.
- 4 Specify the type of object. The object must be a **Deployment**, **DeploymentConfig/dc**, **ReplicaSet/rs**, **ReplicationController/rc**, or **StatefulSet**.
- 5 Specify the name of the object to scale. The object must exist.
- 6 Specify the minimum number of replicas when scaling down.
- 7 Specify the maximum number of replicas when scaling up.
- 8 Use the **metrics** parameter for memory use.
- 9 Specify **cpu** for CPU usage.
- 10 Set to **AverageValue**.
- 11 Set to **averageValue** with the targeted CPU value.

2. Create the horizontal pod autoscaler:

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

Verification

- Check that the horizontal pod autoscaler was created:

```
$ oc get hpa cpu-autoscale
```

Example output

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS
AGE					
cpu-autoscale	Deployment/example	173m/500m	1	10	1
					20m

2.4.6.3. Creating a horizontal pod autoscaler object for a percent of memory use

Using the OpenShift Container Platform CLI, you can create a horizontal pod autoscaler (HPA) to automatically scale an existing object based on a percent of memory use. The HPA scales the pods associated with that object to maintain the memory use that you specify.



NOTE

Use a **Deployment** object or **ReplicaSet** object unless you need a specific feature or behavior provided by other objects.

You can specify the minimum and maximum number of pods and the average memory use that your pods should target. If you do not specify a minimum, the pods are given default values from the OpenShift Container Platform server.

Prerequisites

To use horizontal pod autoscalers, your cluster administrator must have properly configured cluster metrics. You can use the **oc describe PodMetrics <pod-name>** command to determine if metrics are configured. If metrics are configured, the output appears similar to the following, with **Cpu** and **Memory** displayed under **Usage**.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

Example output

```
Name:      openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace: openshift-kube-scheduler
Labels:    <none>
Annotations: <none>
API Version: metrics.k8s.io/v1beta1
Containers:
  Name: wait-for-host-port
  Usage:
    Memory: 0
  Name: scheduler
  Usage:
    Cpu: 8m
    Memory: 45440Ki
Kind: PodMetrics
Metadata:
  Creation Timestamp: 2019-05-23T18:47:56Z
  Self Link: /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-scheduler/pods/openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
  Timestamp: 2019-05-23T18:47:56Z
  Window: 1m0s
  Events: <none>
```

Procedure

1. Create a **HorizontalPodAutoscaler** object similar to the following for an existing object:

```
apiVersion: autoscaling/v2 1
kind: HorizontalPodAutoscaler
metadata:
  name: memory-autoscale 2
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1 3
```

```

kind: Deployment 4
name: example 5
minReplicas: 1 6
maxReplicas: 10 7
metrics: 8
- type: Resource
  resource:
    name: memory 9
    target:
      type: Utilization 10
      averageUtilization: 50 11
behavior: 12
scaleUp:
  stabilizationWindowSeconds: 180
policies:
- type: Pods
  value: 6
  periodSeconds: 120
- type: Percent
  value: 10
  periodSeconds: 120
selectPolicy: Max

```

- 1 Use the **autoscaling/v2** API.
- 2 Specify a name for this horizontal pod autoscaler object.
- 3 Specify the API version of the object to scale:
 - For a ReplicationController, use **v1**.
 - For a DeploymentConfig, use **apps.openshift.io/v1**.
 - For a Deployment, ReplicaSet, Statefulset object, use **apps/v1**.
- 4 Specify the type of object. The object must be a **Deployment**, **DeploymentConfig**, **ReplicaSet**, **ReplicationController**, or **StatefulSet**.
- 5 Specify the name of the object to scale. The object must exist.
- 6 Specify the minimum number of replicas when scaling down.
- 7 Specify the maximum number of replicas when scaling up.
- 8 Use the **metrics** parameter for memory usage.
- 9 Specify **memory** for memory usage.
- 10 Set to **Utilization**.
- 11 Specify **averageUtilization** and a target average memory usage over all the pods, represented as a percent of requested memory. The target pods must have memory requests configured.
- 12 Optional: Specify a scaling policy to control the rate of scaling up or down.

2. Create the horizontal pod autoscaler by using a command similar to the following:

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

For example:

```
$ oc create -f hpa.yaml
```

Example output

```
horizontalpodautoscaler.autoscaling/hpa-resource-metrics-memory created
```

Verification

- Check that the horizontal pod autoscaler was created by using a command similar to the following:

```
$ oc get hpa hpa-resource-metrics-memory
```

Example output

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS
REPLICAS AGE				
hpa-resource-metrics-memory	Deployment/example	2441216/500Mi	1	10
20m				1

- Check the details of the horizontal pod autoscaler by using a command similar to the following:

```
$ oc describe hpa hpa-resource-metrics-memory
```

Example output

```
Name: hpa-resource-metrics-memory
Namespace: default
Labels: <none>
Annotations: <none>
CreationTimestamp: Wed, 04 Mar 2020 16:31:37 +0530
Reference: Deployment/example
Metrics: ( current / target )
  resource memory on pods: 2441216 / 500Mi
Min replicas: 1
Max replicas: 10
ReplicationController pods: 1 current / 1 desired
Conditions:
  Type          Status Reason          Message
  ----          -
  AbleToScale   True   ReadyForNewScale   recommended size matches current size
  ScalingActive True   ValidMetricFound   the HPA was able to successfully calculate a
replica count from memory resource
  ScalingLimited False  DesiredWithinRange the desired count is within the acceptable
range
Events:
  Type          Reason          Age          From          Message
```

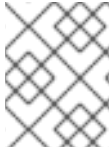
```

Normal SuccessfulRescale 6m34s horizontal-pod-autoscaler New size: 1;
reason: All metrics below target

```

2.4.6.4. Creating a horizontal pod autoscaler object for specific memory use

Using the OpenShift Container Platform CLI, you can create a horizontal pod autoscaler (HPA) to automatically scale an existing object. The HPA scales the pods associated with that object to maintain the average memory use that you specify.



NOTE

Use a **Deployment** object or **ReplicaSet** object unless you need a specific feature or behavior provided by other objects.

You can specify the minimum and maximum number of pods and the average memory use that your pods should target. If you do not specify a minimum, the pods are given default values from the OpenShift Container Platform server.

Prerequisites

To use horizontal pod autoscalers, your cluster administrator must have properly configured cluster metrics. You can use the **oc describe PodMetrics <pod-name>** command to determine if metrics are configured. If metrics are configured, the output appears similar to the following, with **Cpu** and **Memory** displayed under **Usage**.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

Example output

```

Name:      openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace: openshift-kube-scheduler
Labels:    <none>
Annotations: <none>
API Version: metrics.k8s.io/v1beta1
Containers:
  Name: wait-for-host-port
  Usage:
    Memory: 0
  Name: scheduler
  Usage:
    Cpu: 8m
    Memory: 45440Ki
Kind:      PodMetrics
Metadata:
  Creation Timestamp: 2019-05-23T18:47:56Z
  Self Link:         /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-scheduler/pods/openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
  Timestamp:         2019-05-23T18:47:56Z
  Window:            1m0s
  Events:            <none>

```

Procedure

1. Create a **HorizontalPodAutoscaler** object similar to the following for an existing object:

```

apiVersion: autoscaling/v2 ❶
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-resource-metrics-memory ❷
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1 ❸
    kind: Deployment ❹
    name: example ❺
  minReplicas: 1 ❻
  maxReplicas: 10 ❼
  metrics: ❸
  - type: Resource
    resource:
      name: memory ❾
      target:
        type: AverageValue ❿
        averageValue: 500Mi ⓫
  behavior: ❿
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
        - type: Pods
          value: 4
          periodSeconds: 60
        - type: Percent
          value: 10
          periodSeconds: 60
      selectPolicy: Max

```

- ❶ Use the **autoscaling/v2** API.
- ❷ Specify a name for this horizontal pod autoscaler object.
- ❸ Specify the API version of the object to scale:
 - For a **Deployment**, **ReplicaSet**, or **Statefulset** object, use **apps/v1**.
 - For a **ReplicationController**, use **v1**.
 - For a **DeploymentConfig**, use **apps.openshift.io/v1**.
- ❹ Specify the type of object. The object must be a **Deployment**, **DeploymentConfig**, **ReplicaSet**, **ReplicationController**, or **StatefulSet**.
- ❺ Specify the name of the object to scale. The object must exist.
- ❻ Specify the minimum number of replicas when scaling down.
- ❼ Specify the maximum number of replicas when scaling up.

- 8 Use the **metrics** parameter for memory usage.
- 9 Specify **memory** for memory usage.
- 10 Set the type to **AverageValue**.
- 11 Specify **averageValue** and a specific memory value.
- 12 Optional: Specify a scaling policy to control the rate of scaling up or down.

2. Create the horizontal pod autoscaler by using a command similar to the following:

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

For example:

```
$ oc create -f hpa.yaml
```

Example output

```
horizontalpodautoscaler.autoscaling/hpa-resource-metrics-memory created
```

Verification

- Check that the horizontal pod autoscaler was created by using a command similar to the following:

```
$ oc get hpa hpa-resource-metrics-memory
```

Example output

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS
hpa-resource-metrics-memory	Deployment/example	2441216/500Mi	1	10
REPLICAS AGE				
20m				1

- Check the details of the horizontal pod autoscaler by using a command similar to the following:

```
$ oc describe hpa hpa-resource-metrics-memory
```

Example output

```
Name: hpa-resource-metrics-memory
Namespace: default
Labels: <none>
Annotations: <none>
CreationTimestamp: Wed, 04 Mar 2020 16:31:37 +0530
Reference: Deployment/example
Metrics: ( current / target )
  resource memory on pods: 2441216 / 500Mi
Min replicas: 1
Max replicas: 10
```

```
ReplicationController pods: 1 current / 1 desired
```

```
Conditions:
```

Type	Status	Reason	Message
AbleToScale	True	ReadyForNewScale	recommended size matches current size
ScalingActive	True	ValidMetricFound	the HPA was able to successfully calculate a replica count from memory resource
ScalingLimited	False	DesiredWithinRange	the desired count is within the acceptable range

```
Events:
```

Type	Reason	Age	From	Message
Normal	SuccessfulRescale	6m34s	horizontal-pod-autoscaler	New size: 1; reason: All metrics below target

2.4.7. Understanding horizontal pod autoscaler status conditions by using the CLI

You can use the status conditions set to determine whether or not the horizontal pod autoscaler (HPA) is able to scale and whether or not it is currently restricted in any way.

The HPA status conditions are available with the **v2** version of the autoscaling API.

The HPA responds with the following status conditions:

- The **AbleToScale** condition indicates whether HPA is able to fetch and update metrics, as well as whether any backoff-related conditions could prevent scaling.
 - A **True** condition indicates scaling is allowed.
 - A **False** condition indicates scaling is not allowed for the reason specified.
- The **ScalingActive** condition indicates whether the HPA is enabled (for example, the replica count of the target is not zero) and is able to calculate desired metrics.
 - A **True** condition indicates metrics is working properly.
 - A **False** condition generally indicates a problem with fetching metrics.
- The **ScalingLimited** condition indicates that the desired scale was capped by the maximum or minimum of the horizontal pod autoscaler.
 - A **True** condition indicates that you need to raise or lower the minimum or maximum replica count in order to scale.
 - A **False** condition indicates that the requested scaling is allowed.

```
$ oc describe hpa cm-test
```

Example output

```
Name:          cm-test
Namespace:     prom
Labels:        <none>
Annotations:   <none>
CreationTimestamp:  Fri, 16 Jun 2017 18:09:22 +0000
```

```

Reference:          ReplicationController/cm-test
Metrics:            ( current / target )
"http_requests" on pods: 66m / 500m
Min replicas:       1
Max replicas:       4
ReplicationController pods: 1 current / 1 desired
Conditions: 1
  Type      Status Reason          Message
  ----      -
  AbleToScale True    ReadyForNewScale the last scale time was sufficiently old
as to warrant a new scale
  ScalingActive True    ValidMetricFound the HPA was able to successfully
calculate a replica count from pods metric http_request
  ScalingLimited False   DesiredWithinRange the desired replica count is within the
acceptable range
Events:

```

1 The horizontal pod autoscaler status messages.

The following is an example of a pod that is unable to scale:

Example output

```

Conditions:
  Type      Status Reason          Message
  ----      -
  AbleToScale False   FailedGetScale the HPA controller was unable to get the target's current
scale: no matches for kind "ReplicationController" in group "apps"
Events:
  Type      Reason          Age          From          Message
  ----      -
  Warning    FailedGetScale 6s (x3 over 36s) horizontal-pod-autoscaler no matches for kind
"ReplicationController" in group "apps"

```

The following is an example of a pod that could not obtain the needed metrics for scaling:

Example output

```

Conditions:
  Type      Status Reason          Message
  ----      -
  AbleToScale True    SucceededGetScale the HPA controller was able to get the target's
current scale
  ScalingActive False   FailedGetResourceMetric the HPA was unable to compute the replica
count: failed to get cpu utilization: unable to get metrics for resource cpu: no metrics returned from
resource metrics API

```

The following is an example of a pod where the requested autoscaling was less than the required minimums:

Example output

```

Conditions:
  Type      Status Reason          Message

```



```

----      -----      -----      -----
AbleToScale    True    ReadyForNewScale    the last scale time was sufficiently old as to warrant
a new scale
ScalingActive   True    ValidMetricFound    the HPA was able to successfully calculate a replica
count from pods metric http_request
ScalingLimited  False   DesiredWithinRange    the desired replica count is within the acceptable
range

```

2.4.7.1. Viewing horizontal pod autoscaler status conditions by using the CLI

You can view the status conditions set on a pod by the horizontal pod autoscaler (HPA).



NOTE

The horizontal pod autoscaler status conditions are available with the **v2** version of the autoscaling API.

Prerequisites

To use horizontal pod autoscalers, your cluster administrator must have properly configured cluster metrics. You can use the **oc describe PodMetrics <pod-name>** command to determine if metrics are configured. If metrics are configured, the output appears similar to the following, with **Cpu** and **Memory** displayed under **Usage**.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

Example output

```

Name:      openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace: openshift-kube-scheduler
Labels:    <none>
Annotations: <none>
API Version: metrics.k8s.io/v1beta1
Containers:
  Name: wait-for-host-port
  Usage:
    Memory: 0
  Name: scheduler
  Usage:
    Cpu: 8m
    Memory: 45440Ki
Kind: PodMetrics
Metadata:
  Creation Timestamp: 2019-05-23T18:47:56Z
  Self Link: /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-scheduler/pods/openshift-
kube-scheduler-ip-10-0-135-131.ec2.internal
  Timestamp: 2019-05-23T18:47:56Z
  Window: 1m0s
  Events: <none>

```

Procedure

To view the status conditions on a pod, use the following command with the name of the pod:

```
$ oc describe hpa <pod-name>
```

For example:

```
$ oc describe hpa cm-test
```

The conditions appear in the **Conditions** field in the output.

Example output

```
Name:          cm-test
Namespace:     prom
Labels:        <none>
Annotations:   <none>
CreationTimestamp:  Fri, 16 Jun 2017 18:09:22 +0000
Reference:     ReplicationController/cm-test
Metrics:       ( current / target )
"http_requests" on pods:  66m / 500m
Min replicas:    1
Max replicas:    4
ReplicationController pods:  1 current / 1 desired
Conditions: 1
  Type           Status  Reason             Message
  ----           -
  AbleToScale    True    ReadyForNewScale   the last scale time was sufficiently old as to warrant
a new scale
  ScalingActive  True    ValidMetricFound   the HPA was able to successfully calculate a replica
count from pods metric http_request
  ScalingLimited False   DesiredWithinRange the desired replica count is within the acceptable
range
```

2.4.8. Additional resources

- [Understanding deployments and deployment configs](#)
- [Horizontal Pod Autoscaling of Quarkus Application Based on Memory Utilization](#)

2.5. AUTOMATICALLY ADJUST POD RESOURCE LEVELS WITH THE VERTICAL POD AUTOSCALER

The OpenShift Container Platform Vertical Pod Autoscaler Operator (VPA) automatically reviews the historic and current CPU and memory resources for containers in pods. The VPA can update the resource limits and requests based on the usage values it learns. By using individual custom resources (CR), the VPA updates all the pods in a project associated with any built-in workload objects. This includes the following list of object types:

- **Deployment**
- **DeploymentConfig**
- **StatefulSet**
- **Job**

- **DaemonSet**
- **ReplicaSet**
- **ReplicationController**

The VPA can also update certain custom resource object that manage pods. For more information, see [Example custom resources for the Vertical Pod Autoscaler](#) .

The VPA helps you to understand the optimal CPU and memory usage for your pods and can automatically maintain pod resources through the pod lifecycle.

2.5.1. About the Vertical Pod Autoscaler Operator

The Vertical Pod Autoscaler Operator (VPA) is implemented as an API resource and a custom resource (CR). The CR determines the actions for the VPA to take with the pods associated with a specific workload object, such as a daemon set, replication controller, and so forth, in a project.

The VPA consists of three components, each of which has its own pod in the VPA namespace:

Recommender

The VPA recommender monitors the current and past resource consumption. Based on this data, the VPA recommender determines the optimal CPU and memory resources for the pods in the associated workload object.

Updater

The VPA updater checks if the pods in the associated workload object have the correct resources. If the resources are correct, the updater takes no action. If the resources are not correct, the updater kills the pod so that pods' controllers can re-create them with the updated requests.

Admission controller

The VPA admission controller sets the correct resource requests on each new pod in the associated workload object. This applies whether the pod is new or the controller re-created the pod due to the VPA updater actions.

You can use the default recommender or use your own alternative recommender to autoscale based on your own algorithms.

The default recommender automatically computes historic and current CPU and memory usage for the containers in those pods. The default recommender uses this data to determine optimized resource limits and requests to ensure that these pods are operating efficiently at all times. For example, the default recommender suggests reduced resources for pods that are requesting more resources than they are using and increased resources for pods that are not requesting enough.

The VPA then automatically deletes any pods that are out of alignment with these recommendations one at a time, so that your applications can continue to serve requests with no downtime. The workload objects then redeploy the pods with the original resource limits and requests. The VPA uses a mutating admission webhook to update the pods with optimized resource limits and requests before admitting the pods to a node. If you do not want the VPA to delete pods, you can view the VPA resource limits and requests and manually update the pods as needed.

**NOTE**

By default, workload objects must specify a minimum of two replicas for the VPA to automatically delete their pods. Workload objects that specify fewer replicas than this minimum are not deleted. If you manually delete these pods, when the workload object redeploys the pods, the VPA updates the new pods with its recommendations. You can change this minimum by modifying the **VerticalPodAutoscalerController** object as shown in *Changing the VPA minimum value*.

For example, if you have a pod that uses 50% of the CPU but only requests 10%, the VPA determines that the pod is consuming more CPU than requested and deletes the pod. The workload object, such as replica set, restarts the pods and the VPA updates the new pod with its recommended resources.

For developers, you can use the VPA to help ensure that your pods active during periods of high demand by scheduling pods onto nodes that have appropriate resources for each pod.

Administrators can use the VPA to better use cluster resources, such as preventing pods from reserving more CPU resources than needed. The VPA monitors the resources that workloads are actually using and adjusts the resource requirements so capacity is available to other workloads. The VPA also maintains the ratios between limits and requests specified in the initial container configuration.

**NOTE**

If you stop running the VPA or delete a specific VPA CR in your cluster, the resource requests for the pods already modified by the VPA do not change. However, any new pods get the resources defined in the workload object, not the previous recommendations made by the VPA.

2.5.2. Installing the Vertical Pod Autoscaler Operator

You can use the OpenShift Container Platform web console to install the Vertical Pod Autoscaler Operator (VPA).

Procedure

1. In the OpenShift Container Platform web console, click **Ecosystem** → **Software Catalog**.
2. Choose **VerticalPodAutoscaler** from the list of available Operators, and click **Install**.
3. On the **Install Operator** page, ensure that the **Operator recommended namespace** option is selected. This installs the Operator in the mandatory **openshift-vertical-pod-autoscaler** namespace, which is automatically created if it does not exist.
4. Click **Install**.

Verification

1. Verify the installation by listing the VPA components:
 - a. Navigate to **Workloads** → **Pods**.
 - b. Select the **openshift-vertical-pod-autoscaler** project from the drop-down menu and verify that there are four pods running.
 - c. Navigate to **Workloads** → **Deployments** to verify that there are four deployments running.

- Optional: Verify the installation in the OpenShift Container Platform CLI using the following command:

```
$ oc get all -n openshift-vertical-pod-autoscaler
```

The output shows four pods and four deployments:

Example output

```
NAME                                READY STATUS RESTARTS AGE
pod/vertical-pod-autoscaler-operator-85b4569c47-2gmhc 1/1 Running 0      3m13s
pod/vpa-admission-plugin-default-67644fc87f-xq7k9      1/1 Running 0      2m56s
pod/vpa-recommender-default-7c54764b59-8gckt           1/1 Running 0      2m56s
pod/vpa-updater-default-7f6cc87858-47vw9              1/1 Running 0      2m56s

NAME          TYPE      CLUSTER-IP   EXTERNAL-IP  PORT(S)  AGE
service/vpa-webhook ClusterIP  172.30.53.206 <none>      443/TCP  2m56s

NAME                                READY UP-TO-DATE AVAILABLE AGE
deployment.apps/vertical-pod-autoscaler-operator 1/1 1      1      3m13s
deployment.apps/vpa-admission-plugin-default     1/1 1      1      2m56s
deployment.apps/vpa-recommender-default          1/1 1      1      2m56s
deployment.apps/vpa-updater-default              1/1 1      1      2m56s

NAME                                DESIRED CURRENT READY AGE
replicaset.apps/vertical-pod-autoscaler-operator-85b4569c47 1      1      1      3m13s
replicaset.apps/vpa-admission-plugin-default-67644fc87f      1      1      1      2m56s
replicaset.apps/vpa-recommender-default-7c54764b59           1      1      1      2m56s
replicaset.apps/vpa-updater-default-7f6cc87858              1      1      1      2m56s
```

2.5.3. Moving the Vertical Pod Autoscaler Operator components

The Vertical Pod Autoscaler Operator (VPA) and each component has its own pod in the VPA namespace on the control plane nodes. You can move the VPA Operator and component pods to infrastructure or worker nodes by adding a node selector to the VPA subscription and the **VerticalPodAutoscalerController** CR.

You can create and use infrastructure nodes to host only infrastructure components. For example, the default router, the integrated container image registry, and the components for cluster metrics and monitoring. These infrastructure nodes are not counted toward the total number of subscriptions that are required to run the environment. For more information, see *Creating infrastructure machine sets*.

You can move the components to the same node or separate nodes as appropriate for your organization.

The following example shows the default deployment of the VPA pods to the control plane nodes.

Example output

```
NAME                                READY STATUS RESTARTS AGE IP      NODE
NOMINATED NODE READINESS GATES
vertical-pod-autoscaler-operator-6c75fcc9cd-5pb6z 1/1 Running 0      7m59s 10.128.2.24
c416-tfsbj-master-1 <none> <none>
vpa-admission-plugin-default-6cb78d6f8b-rpcrj      1/1 Running 0      5m37s 10.129.2.22
c416-tfsbj-master-1 <none> <none>
```

```

vpa-recommender-default-66846bd94c-dsmpp      1/1   Running   0       5m37s  10.129.2.20
c416-tfsbj-master-0 <none>          <none>
vpa-updater-default-db8b58df-2nkvf           1/1   Running   0       5m37s  10.129.2.21  c416-
tfsbj-master-1 <none>          <none>

```

Procedure

1. Move the VPA Operator pod by adding a node selector to the **Subscription** custom resource (CR) for the VPA Operator:

- a. Edit the CR:

```
$ oc edit Subscription vertical-pod-autoscaler -n openshift-vertical-pod-autoscaler
```

- b. Add a node selector to match the node role label on the node where you want to install the VPA Operator pod:

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  labels:
    operators.coreos.com/vertical-pod-autoscaler.openshift-vertical-pod-autoscaler: ""
  name: vertical-pod-autoscaler
# ...
spec:
  config:
    nodeSelector:
      node-role.kubernetes.io/<node_role>: "" 1

```

- 1 1 Specifies the node role of the node where you want to move the VPA Operator pod.

NOTE

If the infra node uses taints, you need to add a toleration to the **Subscription** CR.

For example:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  labels:
    operators.coreos.com/vertical-pod-autoscaler.openshift-vertical-pod-
autoscaler: ""
  name: vertical-pod-autoscaler
# ...
spec:
  config:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations: 1
      - key: "node-role.kubernetes.io/infra"
        operator: "Exists"
        effect: "NoSchedule"
```

- 1** Specifies a toleration for a taint on the node where you want to move the VPA Operator pod.

2. Move each VPA component by adding node selectors to the **VerticalPodAutoscaler** custom resource (CR):

- a. Edit the CR:

```
$ oc edit VerticalPodAutoscalerController default -n openshift-vertical-pod-autoscaler
```

- b. Add node selectors to match the node role label on the node where you want to install the VPA components:

```
apiVersion: autoscaling.openshift.io/v1
kind: VerticalPodAutoscalerController
metadata:
  name: default
  namespace: openshift-vertical-pod-autoscaler
# ...
spec:
  deploymentOverrides:
    admission:
      container:
        resources: {}
    nodeSelector:
      node-role.kubernetes.io/<node_role>: "" 1
  recommender:
    container:
      resources: {}
    nodeSelector:
```

```
node-role.kubernetes.io/<node_role>: "" 2
updater:
  container:
    resources: {}
  nodeSelector:
    node-role.kubernetes.io/<node_role>: "" 3
```

- 1 Optional: Specifies the node role for the VPA admission pod.
- 2 Optional: Specifies the node role for the VPA recommender pod.
- 3 Optional: Specifies the node role for the VPA updater pod.

NOTE

If a target node uses taints, you need to add a toleration to the **VerticalPodAutoscalerController** CR.

For example:

```
apiVersion: autoscaling.openshift.io/v1
kind: VerticalPodAutoscalerController
metadata:
  name: default
  namespace: openshift-vertical-pod-autoscaler
# ...
spec:
  deploymentOverrides:
    admission:
      container:
        resources: {}
        nodeSelector:
          node-role.kubernetes.io/worker: ""
        tolerations: ❶
        - key: "my-example-node-taint-key"
          operator: "Exists"
          effect: "NoSchedule"
    recommender:
      container:
        resources: {}
        nodeSelector:
          node-role.kubernetes.io/worker: ""
        tolerations: ❷
        - key: "my-example-node-taint-key"
          operator: "Exists"
          effect: "NoSchedule"
    updater:
      container:
        resources: {}
        nodeSelector:
          node-role.kubernetes.io/worker: ""
        tolerations: ❸
        - key: "my-example-node-taint-key"
          operator: "Exists"
          effect: "NoSchedule"
```

- ❶ Specifies a toleration for the admission controller pod for a taint on the node where you want to install the pod.
- ❷ Specifies a toleration for the recommender pod for a taint on the node where you want to install the pod.
- ❸ Specifies a toleration for the updater pod for a taint on the node where you want to install the pod.

Verification

- You can verify the pods have moved by using the following command:

```
$ oc get pods -n openshift-vertical-pod-autoscaler -o wide
```

The pods are no longer deployed to the control plane nodes. In the following example output, the node is now an infra node, not a control plane node.

Example output

```
NAME                                READY STATUS  RESTARTS  AGE   IP
NODE                                NOMINATED NODE READINESS GATES
vertical-pod-autoscaler-operator-6c75fcc9cd-5pb6z 1/1   Running  0         7m59s
10.128.2.24 c416-tfsbj-infra-eastus3-2bndt <none> <none>
vpa-admission-plugin-default-6cb78d6f8b-rpcrj 1/1   Running  0         5m37s
10.129.2.22 c416-tfsbj-infra-eastus1-lrgj8 <none> <none>
vpa-recommender-default-66846bd94c-dsmpp 1/1   Running  0         5m37s
10.129.2.20 c416-tfsbj-infra-eastus1-lrgj8 <none> <none>
vpa-updater-default-db8b58df-2nkvf 1/1   Running  0         5m37s 10.129.2.21
c416-tfsbj-infra-eastus1-lrgj8 <none> <none>
```

2.5.4. About using the Vertical Pod Autoscaler Operator

To use the Vertical Pod Autoscaler Operator (VPA), you create a VPA custom resource (CR) for a workload object in your cluster. The VPA learns and applies the optimal CPU and memory resources for the pods associated with that workload object. You can use a VPA with a deployment, stateful set, job, daemon set, replica set, or replication controller workload object. The VPA CR must be in the same project as the pods that you want to check.

You use the VPA CR to associate a workload object and specify the mode that the VPA operates in:

- The **Auto** and **Recreate** modes automatically apply the VPA CPU and memory recommendations throughout the pod lifetime. The VPA deletes any pods in the project that are out of alignment with its recommendations. When redeployed by the workload object, the VPA updates the new pods with its recommendations.
- The **Initial** mode automatically applies VPA recommendations only at pod creation.
- The **Off** mode only provides recommended resource limits and requests. You can then manually apply the recommendations. The **Off** mode does not update pods.

You can also use the CR to opt-out certain containers from VPA evaluation and updates.

For example, a pod has the following limits and requests:

```
resources:
  limits:
    cpu: 1
    memory: 500Mi
  requests:
    cpu: 500m
    memory: 100Mi
```

After creating a VPA that is set to **Auto**, the VPA learns the resource usage and deletes the pod. When redeployed, the pod uses the new resource limits and requests:

-

```
resources:
  limits:
    cpu: 50m
    memory: 1250Mi
  requests:
    cpu: 25m
    memory: 262144k
```

You can view the VPA recommendations by using the following command:

```
$ oc get vpa <vpa-name> --output yaml
```

After a few minutes, the output shows the recommendations for CPU and memory requests, similar to the following:

Example output

```
...
status:
...
  recommendation:
    containerRecommendations:
      - containerName: frontend
        lowerBound:
          cpu: 25m
          memory: 262144k
        target:
          cpu: 25m
          memory: 262144k
        uncappedTarget:
          cpu: 25m
          memory: 262144k
        upperBound:
          cpu: 262m
          memory: "274357142"
      - containerName: backend
        lowerBound:
          cpu: 12m
          memory: 131072k
        target:
          cpu: 12m
          memory: 131072k
        uncappedTarget:
          cpu: 12m
          memory: 131072k
        upperBound:
          cpu: 476m
          memory: "498558823"
    ...
```

The output shows the recommended resources, **target**, the minimum recommended resources, **lowerBound**, the highest recommended resources, **upperBound**, and the most recent resource recommendations, **uncappedTarget**.

The VPA uses the **lowerBound** and **upperBound** values to determine if a pod needs updating. If a pod has resource requests less than the **lowerBound** values or more than the **upperBound** values, the VPA terminates and recreates the pod with the **target** values.

2.5.4.1. Changing the VPA minimum value

By default, workload objects must specify a minimum of two replicas in order for the VPA to automatically delete and update their pods. As a result, workload objects that specify fewer than two replicas are not automatically acted upon by the VPA. The VPA does update new pods from these workload objects if a process external to the VPA restarts the pods. You can change this cluster-wide minimum value by modifying the **minReplicas** parameter in the **VerticalPodAutoscalerController** custom resource (CR).

For example, if you set **minReplicas** to **3**, the VPA does not delete and update pods for workload objects that specify fewer than three replicas.



NOTE

If you set **minReplicas** to **1**, the VPA can delete the only pod for a workload object that specifies only one replica. Use this setting with one-replica objects only if your workload can tolerate downtime whenever the VPA deletes a pod to adjust its resources. To avoid unwanted downtime with one-replica objects, configure the VPA CRs with the **podUpdatePolicy** set to **Initial**, which automatically updates the pod only when a process external to the VPA restarts, or **Off**, which you can use to update the pod manually at an appropriate time for your application.

Example VerticalPodAutoscalerController object

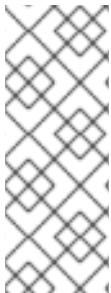
```
apiVersion: autoscaling.openshift.io/v1
kind: VerticalPodAutoscalerController
metadata:
  creationTimestamp: "2021-04-21T19:29:49Z"
  generation: 2
  name: default
  namespace: openshift-vertical-pod-autoscaler
  resourceVersion: "142172"
  uid: 180e17e9-03cc-427f-9955-3b4d7aeb2d59
spec:
  minReplicas: 3 1
  podMinCPUMillicores: 25
  podMinMemoryMb: 250
  recommendationOnly: false
  safetyMarginFraction: 0.15
```

- ¹ Specify the minimum number of replicas in a workload object for the VPA to act on. Any objects with replicas fewer than the minimum are not automatically deleted by the VPA.

2.5.4.2. Automatically applying VPA recommendations

To use the VPA to automatically update pods, create a VPA CR for a specific workload object with **updateMode** set to **Auto** or **Recreate**.

When the pods are created for the workload object, the VPA constantly monitors the containers to analyze their CPU and memory needs. The VPA deletes any pods that do not meet the VPA recommendations for CPU and memory. When redeployed, the pods use the new resource limits and requests based on the VPA recommendations, honoring any pod disruption budget set for your applications. The recommendations are added to the **status** field of the VPA CR for reference.



NOTE

By default, workload objects must specify a minimum of two replicas in order for the VPA to automatically delete their pods. Workload objects that specify fewer replicas than this minimum are not deleted. If you manually delete these pods, when the workload object redeployes the pods, the VPA does update the new pods with its recommendations. You can change this minimum by modifying the **VerticalPodAutoscalerController** object as shown in *Changing the VPA minimum value*.

Example VPA CR for the **Auto** mode

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment ❶
    name: frontend ❷
  updatePolicy:
    updateMode: "Auto" ❸
```

- ❶ The type of workload object you want this VPA CR to manage.
- ❷ The name of the workload object you want this VPA CR to manage.
- ❸ Set the mode to **Auto** or **Recreate**:
 - **Auto**. The VPA assigns resource requests on pod creation and updates the existing pods by terminating them when the requested resources differ significantly from the new recommendation.
 - **Recreate**. The VPA assigns resource requests on pod creation and updates the existing pods by terminating them when the requested resources differ significantly from the new recommendation. Use this mode rarely, only if you need to ensure that when the resource request changes the pods restart.

**NOTE**

Before a VPA can determine recommendations for resources and apply the recommended resources to new pods, operating pods must exist and be running in the project.

If a workload's resource usage, such as CPU and memory, is consistent, the VPA can determine recommendations for resources in a few minutes. If a workload's resource usage is inconsistent, the VPA must collect metrics at various resource usage intervals for the VPA to make an accurate recommendation.

2.5.4.3. Automatically applying VPA recommendations on pod creation

To use the VPA to apply the recommended resources only when a pod is first deployed, create a VPA CR for a specific workload object with **updateMode** set to **Initial**.

Then, manually delete any pods associated with the workload object that you want to use the VPA recommendations. In the **Initial** mode, the VPA does not delete pods and does not update the pods as it learns new resource recommendations.

Example VPA CR for the Initial mode

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment 1
    name: frontend 2
  updatePolicy:
    updateMode: "Initial" 3
```

- 1** The type of workload object you want this VPA CR to manage.
- 2** The name of the workload object you want this VPA CR to manage.
- 3** Set the mode to **Initial**. The VPA assigns resources when pods are created and does not change the resources during the lifetime of the pod.

**NOTE**

Before a VPA can determine recommended resources and apply the recommendations to new pods, operating pods must exist and be running in the project.

To obtain the most accurate recommendations from the VPA, wait at least 8 days for the pods to run and for the VPA to stabilize.

2.5.4.4. Manually applying VPA recommendations

To use the VPA to only determine the recommended CPU and memory values, create a VPA CR for a specific workload object with **updateMode** set to **Off**.

When the pods are created for that workload object, the VPA analyzes the CPU and memory needs of the containers and records those recommendations in the **status** field of the VPA CR. The VPA does not update the pods as it determines new resource recommendations.

Example VPA CR for the Off mode

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment ❶
    name: frontend ❷
  updatePolicy:
    updateMode: "Off" ❸
```

- ❶ The type of workload object you want this VPA CR to manage.
- ❷ The name of the workload object you want this VPA CR to manage.
- ❸ Set the mode to **Off**.

You can view the recommendations by using the following command.

```
$ oc get vpa <vpa-name> --output yaml
```

With the recommendations, you can edit the workload object to add CPU and memory requests, then delete and redeploy the pods by using the recommended resources.



NOTE

Before a VPA can determine recommended resources and apply the recommendations to new pods, operating pods must exist and be running in the project.

To obtain the most accurate recommendations from the VPA, wait at least 8 days for the pods to run and for the VPA to stabilize.

2.5.4.5. Exempting containers from applying VPA recommendations

If your workload object has multiple containers and you do not want the VPA to evaluate and act on all of the containers, create a VPA CR for a specific workload object and add a **resourcePolicy** to opt-out specific containers.

When the VPA updates the pods with recommended resources, any containers with a **resourcePolicy** are not updated and the VPA does not present recommendations for those containers in the pod.

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
spec:
```

```

targetRef:
  apiVersion: "apps/v1"
  kind:      Deployment 1
  name:      frontend 2
updatePolicy:
  updateMode: "Auto" 3
resourcePolicy: 4
containerPolicies:
- containerName: my-opt-sidecar
  mode: "Off"

```

- 1** The type of workload object you want this VPA CR to manage.
- 2** The name of the workload object you want this VPA CR to manage.
- 3** Set the mode to **Auto**, **Recreate**, **Initial**, or **Off**. Use the **Recreate** mode rarely, only if you need to ensure that when the resource request changes the pods restart.
- 4** Specify the containers that you do not want updated by the VPA and set the **mode** to **Off**.

For example, a pod has two containers, the same resource requests and limits:

```

# ...
spec:
  containers:
  - name: frontend
    resources:
      limits:
        cpu: 1
        memory: 500Mi
      requests:
        cpu: 500m
        memory: 100Mi
  - name: backend
    resources:
      limits:
        cpu: "1"
        memory: 500Mi
      requests:
        cpu: 500m
        memory: 100Mi
# ...

```

After launching a VPA CR with the **backend** container set to opt-out, the VPA terminates and recreates the pod with the recommended resources applied only to the **frontend** container:

```

...
spec:
  containers:
  name: frontend
  resources:
    limits:
      cpu: 50m
      memory: 1250Mi

```



```

requests:
  cpu: 25m
  memory: 262144k
...
name: backend
resources:
  limits:
    cpu: "1"
    memory: 500Mi
  requests:
    cpu: 500m
    memory: 100Mi
...

```

2.5.4.6. Performance tuning the VPA Operator

As a cluster administrator, you can tune the performance of your Vertical Pod Autoscaler Operator (VPA) to limit the rate at which the VPA makes requests of the Kubernetes API server and to specify the CPU and memory resources for the VPA recommender, updater, and admission controller component pods.

You can also configure the VPA to monitor only those workloads a VPA custom resource (CR) manages. By default, the VPA monitors every workload in the cluster. As a result, the VPA accrues and stores 8 days of historical data for all workloads. This can be used by the VPA if a new VPA CR is created for a workload. However, this causes the VPA to use significant CPU and memory. This can cause the VPA to fail, particularly on larger clusters. By configuring the VPA to monitor only workloads with a VPA CR, you can save on CPU and memory resources. One tradeoff is that where you have a running workload and you create a VPA CR to manage that workload. The VPA does not have any historical data for that workload. As a result, the initial recommendations are not as useful as those after the workload is running for some time.

Use these tunings to ensure the VPA has enough resources to operate at peak efficiency and to prevent throttling, and a possible delay in pod admissions.

You can perform the following tunings on the VPA components by editing the **VerticalPodAutoscalerController** custom resource (CR):

- To prevent throttling and pod admission delays, set the queries per second (QPS) and burst rates for VPA requests of the Kubernetes API server by using the **kube-api-qps** and **kube-api-burst** parameters.
- To ensure enough CPU and memory, set the CPU and memory requests for VPA component pods by using the standard **cpu** and **memory** resource requests.
- To configure the VPA to monitor only workloads that the VPA CR manages, set the **memory-saver** parameter to **true** for the recommender component.

For guidelines on the resources and rate limits that you could set for each VPA component, the following tables provide recommended baseline values, depending on the size of your cluster and other factors.



IMPORTANT

These recommended values derive from internal Red Hat testing on clusters that are not necessarily representative of real-world clusters. Before you configure a production cluster, ensure you test these values in a non-production cluster.

Table 2.2. Requests by containers in the cluster

C o m p o n e n t	1-500 contain ers		500- 1,000 contain ers		1,000- 2,000 contain ers		2,000- 4,000 contai ners		4,000+ containers	
	C P U	M e m o r y	C P U	M e m o r y	C P U	M e m o r y	C P U	M e m o r y	CPU	Memory
A d m i s s i o n	2 5 m	5 0 M i	2 5 m	7 5 M i	4 0 m	15 0 M i	7 5 m	2 6 0 M i	$(0.03c)/2 + 10^{[1]}$	$(0.1c)/2 + 50^{[1]}$
R e c o m m e n d e r	2 5 m	1 0 0 M i	5 0 m	16 0 M i	7 5 m	2 7 5 M i	12 0 m	4 2 0 M i	$(0.05c)/2 + 50^{[1]}$	$(0.15c)/2 + 120^{[1]}$
U p d a t e r	2 5 m	1 0 0 M i	5 0 m	2 2 0 M i	8 0 m	3 5 0 M i	1 5 0 m	5 0 0 M i	$(0.07c)/2 + 20^{[1]}$	$(0.15c)/2 + 200^{[1]}$

1. **c** is the number of containers in the cluster.

**NOTE**

It is recommended that you set the memory limit on your containers to at least double the recommended requests in the table. However, because CPU is a compressible resource, setting CPU limits for containers can throttle the VPA. As such, it is recommended that you do not set a CPU limit on your containers.

Table 2.3. Rate limits by VPAs in the cluster

C o m p o n e n t	1-150 VPAs		151-500 VPAs		501-2,000 VPAs		2,001-4,000 VPAs	
	QPS Limit [1]	Burst [2]	QPS Limit	Burst	QPS Limit	Burst	QPS Limit	Burst
R e c o m m e n d e r	5	10	30	60	60	120	120	240
U p d a t e r	5	10	30	60	60	120	120	240

1. QPS specifies the queries per second (QPS) limit when making requests to Kubernetes API server. The default for the updater and recommender pods is **5.0**.
2. Burst specifies the burst limit when making requests to Kubernetes API server. The default for the updater and recommender pods is **10.0**.



NOTE

If you have more than 4,000 VPAs in your cluster, it is recommended that you start performance tuning with the values in the table and slowly increase the values until you achieve the required recommender and updater latency and performance. Adjust these values slowly because increased QPS and Burst can affect cluster health and slow down the Kubernetes API server if too many API requests are sent to the API server from the VPA components.

The following example VPA controller CR is for a cluster with 1,000 to 2,000 containers and a pod creation surge of 26 to 50. The CR sets the following values:

- The container memory and CPU requests for all three VPA components
- The container memory limit for all three VPA components
- The QPS and burst rates for all three VPA components
- The **memory-saver** parameter to **true** for the VPA recommender component

Example VerticalPodAutoscalerController CR

```
apiVersion: autoscaling.openshift.io/v1
kind: VerticalPodAutoscalerController
metadata:
  name: default
  namespace: openshift-vertical-pod-autoscaler
spec:
  deploymentOverrides:
    admission: 1
    container:
      args: 2
      - '--kube-api-qps=50.0'
      - '--kube-api-burst=100.0'
    resources:
      requests: 3
      cpu: 40m
      memory: 150Mi
    limits:
      memory: 300Mi
  recommender: 4
  container:
    args:
      - '--kube-api-qps=60.0'
      - '--kube-api-burst=120.0'
      - '--memory-saver=true' 5
    resources:
      requests:
        cpu: 75m
        memory: 275Mi
      limits:
        memory: 550Mi
  updater: 6
  container:
    args:
```

```

- '--kube-api-qps=60.0'
- '--kube-api-burst=120.0'
resources:
  requests:
    cpu: 80m
    memory: 350M
  limits:
    memory: 700Mi
minReplicas: 2
podMinCPUMillicores: 25
podMinMemoryMb: 250
recommendationOnly: false
safetyMarginFraction: 0.15

```

- 1 Specifies the tuning parameters for the VPA admission controller.
- 2 Specifies the API QPS and burst rates for the VPA admission controller.
 - **kube-api-qps**: Specifies the queries per second (QPS) limit when making requests to Kubernetes API server. The default is **5.0**.
 - **kube-api-burst**: Specifies the burst limit when making requests to Kubernetes API server. The default is **10.0**.
- 3 Specifies the resource requests and limits for the VPA admission controller pod.
- 4 Specifies the tuning parameters for the VPA recommender.
- 5 Specifies that the VPA Operator monitors only workloads with a VPA CR. The default is **false**.
- 6 Specifies the tuning parameters for the VPA updater.

You can verify that the settings were applied to each VPA component pod.

Example updater pod

```

apiVersion: v1
kind: Pod
metadata:
  name: vpa-updater-default-d65ffb9dc-hgw44
  namespace: openshift-vertical-pod-autoscaler
# ...
spec:
  containers:
    - args:
      - --logtostderr
      - --v=1
      - --min-replicas=2
      - --kube-api-qps=60.0
      - --kube-api-burst=120.0
    # ...
  resources:
    requests:

```

```

    cpu: 80m
    memory: 350M
# ...

```

Example admission controller pod

```

apiVersion: v1
kind: Pod
metadata:
  name: vpa-admission-plugin-default-756999448c-l7tsd
  namespace: openshift-vertical-pod-autoscaler
# ...
spec:
  containers:
  - args:
    - --logtostderr
    - --v=1
    - --tls-cert-file=/data/tls-certs/tls.crt
    - --tls-private-key=/data/tls-certs/tls.key
    - --client-ca-file=/data/tls-ca-certs/service-ca.crt
    - --webhook-timeout-seconds=10
    - --kube-api-qps=50.0
    - --kube-api-burst=100.0
# ...
  resources:
    requests:
      cpu: 40m
      memory: 150Mi
# ...

```

Example recommender pod

```

apiVersion: v1
kind: Pod
metadata:
  name: vpa-recommender-default-74c979dbbc-znrd2
  namespace: openshift-vertical-pod-autoscaler
# ...
spec:
  containers:
  - args:
    - --logtostderr
    - --v=1
    - --recommendation-margin-fraction=0.15
    - --pod-recommendation-min-cpu-millicores=25
    - --pod-recommendation-min-memory-mb=250
    - --kube-api-qps=60.0
    - --kube-api-burst=120.0
    - --memory-saver=true
# ...
  resources:
    requests:
      cpu: 75m
      memory: 275Mi
# ...

```

2.5.4.7. Custom memory bump-up after OOM event

If your cluster experiences an OOM (out of memory) event, the Vertical Pod Autoscaler Operator (VPA) increases the memory recommendation. The basis for the recommendation is the memory consumption observed during the OOM event and a specified multiplier value to prevent future crashes due to insufficient memory.

The recommendation is the higher of two calculations: the memory in use by the pod when the OOM event happened multiplied by a specified number of bytes or a specified percentage. The following formula represents the calculation:

```
recommendation = max(memory-usage-in-oom-event + oom-min-bump-up-bytes, memory-usage-in-oom-event * oom-bump-up-ratio)
```

You can configure the memory increase by specifying the following values in the recommender pod:

- **oom-min-bump-up-bytes.** This value, in bytes, is a specific increase in memory after an OOM event occurs. The default is **100MiB**.
- **oom-bump-up-ratio.** This value is a percentage increase in memory when the OOM event occurred. The default value is **1.2**.

For example, if the pod memory usage during an OOM event is 100 MB, and **oom-min-bump-up-bytes** is set to 150 MB with a **oom-min-bump-ratio** of 1.2. After an OOM event, the VPA recommends increasing the memory request for that pod to 150 MB, as it is higher than at 120 MB (100 MB * 1.2).

Example recommender deployment object

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vpa-recommender-default
  namespace: openshift-vertical-pod-autoscaler
# ...
spec:
# ...
  template:
# ...
    spec
      containers:
      - name: recommender
        args:
        - --oom-bump-up-ratio=2.0
        - --oom-min-bump-up-bytes=524288000
# ...
```

Additional resources

- [Understanding OOM kill policy](#)

2.5.4.8. Using an alternative recommender

You can use your own recommender to autoscale based on your own algorithms. If you do not specify an alternative recommender, OpenShift Container Platform uses the default recommender, which suggests CPU and memory requests based on historical usage. Because there is no universal

recommendation policy that applies to all types of workloads, you might want to create and deploy different recommenders for specific workloads.

For example, the default recommender might not accurately predict future resource usage when containers exhibit certain resource behaviors. Examples are cyclical patterns that alternate between usage spikes and idling as used by monitoring applications, or recurring and repeating patterns used with deep learning applications. Using the default recommender with these usage behaviors might result in significant over-provisioning and Out of Memory (OOM) kills for your applications.



NOTE

Instructions for how to create a recommender are beyond the scope of this documentation.

Procedure

To use an alternative recommender for your pods:

1. Create a service account for the alternative recommender and bind that service account to the required cluster role:

```

apiVersion: v1 1
kind: ServiceAccount
metadata:
  name: alt-vpa-recommender-sa
  namespace: <namespace_name>
---
apiVersion: rbac.authorization.k8s.io/v1 2
kind: ClusterRoleBinding
metadata:
  name: system:example-metrics-reader
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:metrics-reader
subjects:
- kind: ServiceAccount
  name: alt-vpa-recommender-sa
  namespace: <namespace_name>
---
apiVersion: rbac.authorization.k8s.io/v1 3
kind: ClusterRoleBinding
metadata:
  name: system:example-vpa-actor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:vpa-actor
subjects:
- kind: ServiceAccount
  name: alt-vpa-recommender-sa
  namespace: <namespace_name>
---
apiVersion: rbac.authorization.k8s.io/v1 4
kind: ClusterRoleBinding

```



```

metadata:
  name: system:example-vpa-target-reader-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:vpa-target-reader
subjects:
- kind: ServiceAccount
  name: alt-vpa-recommender-sa
  namespace: <namespace_name>

```

- 1 Creates a service account for the recommender in the namespace that displays the recommender.
 - 2 Binds the recommender service account to the **metrics-reader** role. Specify the namespace for where to deploy the recommender.
 - 3 Binds the recommender service account to the **vpa-actor** role. Specify the namespace for where to deploy the recommender.
 - 4 Binds the recommender service account to the **vpa-target-reader** role. Specify the namespace for where to display the recommender.
2. To add the alternative recommender to the cluster, create a **Deployment** object similar to the following:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: alt-vpa-recommender
  namespace: <namespace_name>
spec:
  replicas: 1
  selector:
    matchLabels:
      app: alt-vpa-recommender
  template:
    metadata:
      labels:
        app: alt-vpa-recommender
    spec:
      containers: 1
      - name: recommender
        image: quay.io/example/alt-recommender:latest 2
        imagePullPolicy: Always
        resources:
          limits:
            cpu: 200m
            memory: 1000Mi
          requests:
            cpu: 50m
            memory: 500Mi
      ports:
      - name: prometheus
        containerPort: 8942

```

```

securityContext:
  allowPrivilegeEscalation: false
  capabilities:
    drop:
      - ALL
  seccompProfile:
    type: RuntimeDefault
serviceAccountName: alt-vpa-recommender-sa ❸
securityContext:
  runAsNonRoot: true

```

- ❶ Creates a container for your alternative recommender.
- ❷ Specifies your recommender image.
- ❸ Associates the service account that you created for the recommender.

A new pod is created for the alternative recommender in the same namespace.

```
$ oc get pods
```

Example output

NAME	READY	STATUS	RESTARTS	AGE
frontend-845d5478d-558zf	1/1	Running	0	4m25s
frontend-845d5478d-7z9gx	1/1	Running	0	4m25s
frontend-845d5478d-b7l4j	1/1	Running	0	4m25s
vpa-alt-recommender-55878867f9-6tp5v	1/1	Running	0	9s

3. Configure a Vertical Pod Autoscaler Operator (VPA) custom resource (CR) that includes the name of the alternative recommender **Deployment** object.

Example VPA CR to include the alternative recommender

```

apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
  namespace: <namespace_name>
spec:
  recommenders:
    - name: alt-vpa-recommender ❶
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment ❷
    name: frontend

```

- ❶ Specifies the name of the alternative recommender deployment.
- ❷ Specifies the name of an existing workload object you want this VPA to manage.

2.5.5. Using the Vertical Pod Autoscaler Operator

You can use the Vertical Pod Autoscaler Operator (VPA) by creating a VPA custom resource (CR). The CR indicates the pods to analyze and determines the actions for the VPA to take with those pods.

You can use the VPA to scale built-in resources such as deployments or stateful sets, and custom resources that manage pods. For more information, see "About using the Vertical Pod Autoscaler Operator".

Prerequisites

- Ensure the workload object that you want to autoscale exists.
- Ensure that if you want to use an alternative recommender, a deployment including that recommender exists.

Procedure

To create a VPA CR for a specific workload object:

1. Change to the location of the project for the workload object you want to scale.
 - a. Create a VPA CR YAML file:

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment 1
    name: frontend 2
  updatePolicy:
    updateMode: "Auto" 3
  resourcePolicy: 4
    containerPolicies:
      - containerName: my-opt-sidecar
        mode: "Off"
  recommenders: 5
    - name: my-recommender
```

- 1 Specify the type of workload object you want this VPA to manage: **Deployment**, **StatefulSet**, **Job**, **DaemonSet**, **ReplicaSet**, or **ReplicationController**.
- 2 Specify the name of an existing workload object you want this VPA to manage.
- 3 Specify the VPA mode:
 - **Auto** to automatically apply the recommended resources on pods associated with the controller. The VPA terminates existing pods and creates new pods with the recommended resource limits and requests.
 - **Recreate** to automatically apply the recommended resources on pods associated with the workload object. The VPA terminates existing pods and creates new pods with the recommended resource limits and requests. Use the **Recreate** mode rarely, only if you need to ensure that the pods restart whenever the resource request changes.

- **Initial** to automatically apply the recommended resources to newly-created pods associated with the workload object. The VPA does not update the pods as it learns new resource recommendations.
- **Off** to only generate resource recommendations for the pods associated with the workload object. The VPA does not update the pods as it learns new resource recommendations and does not apply the recommendations to new pods.

- 4 Optional. Specify the containers you want to opt-out and set the mode to **Off**.
- 5 Optional. Specify an alternative recommender.

b. Create the VPA CR:

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

After a few moments, the VPA learns the resource usage of the containers in the pods associated with the workload object.

You can view the VPA recommendations by using the following command:

```
$ oc get vpa <vpa-name> --output yaml
```

The output shows the recommendations for CPU and memory requests, similar to the following:

Example output

```
...
status:
...

recommendation:
  containerRecommendations:
    - containerName: frontend
      lowerBound: 1
        cpu: 25m
        memory: 262144k
      target: 2
        cpu: 25m
        memory: 262144k
      uncappedTarget: 3
        cpu: 25m
        memory: 262144k
      upperBound: 4
        cpu: 262m
        memory: "274357142"
    - containerName: backend
      lowerBound:
        cpu: 12m
        memory: 131072k
      target:
        cpu: 12m
```

```

memory: 131072k
uncappedTarget:
  cpu: 12m
  memory: 131072k
upperBound:
  cpu: 476m
  memory: "498558823"
...

```

- 1 **lowerBound** is the minimum recommended resource levels.
- 2 **target** is the recommended resource levels.
- 3 **upperBound** is the highest recommended resource levels.
- 4 **uncappedTarget** is the most recent resource recommendations.

2.5.5.1. Example custom resources for the Vertical Pod Autoscaler

The Vertical Pod Autoscaler Operator (VPA) can update not only built-in resources such as deployments or stateful sets, but also custom resources that manage pods.

To use the VPA with a custom resource when you create the **CustomResourceDefinition** (CRD) object, you must configure the **labelSelectorPath** field in the **/scale** subresource. The **/scale** subresource creates a **Scale** object. The **labelSelectorPath** field defines the JSON path inside the custom resource that corresponds to **status.selector** in the **Scale** object and in the custom resource. The following is an example of a **CustomResourceDefinition** and a **CustomResource** that fulfills these requirements, along with a **VerticalPodAutoscaler** definition that targets the custom resource. The following example shows the **/scale** subresource contract.



NOTE

This example does not result in the VPA scaling pods because there is no controller for the custom resource that allows it to own any pods. As such, you must write a controller in a language supported by Kubernetes to manage the reconciliation and state management between the custom resource and your pods. The example illustrates the configuration for the VPA to understand the custom resource as scalable.

Example custom CRD, CR

```

apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: scalablepods.testing.openshift.io
spec:
  group: testing.openshift.io
  versions:
    - name: v1
      served: true
      storage: true
  schema:
    openAPIV3Schema:

```

```

    type: object
    properties:
      spec:
        type: object
        properties:
          replicas:
            type: integer
            minimum: 0
          selector:
            type: string
        status:
          type: object
          properties:
            replicas:
              type: integer
      subresources:
        status: {}
        scale:
          specReplicasPath: .spec.replicas
          statusReplicasPath: .status.replicas
          labelSelectorPath: .spec.selector 1
    scope: Namespaced
    names:
      plural: scalablepods
      singular: scalablepod
      kind: ScalablePod
      shortNames:
        - spod

```

- 1** Specifies the JSON path that corresponds to **status.selector** field of the custom resource object.

Example custom CR

```

apiVersion: testing.openshift.io/v1
kind: ScalablePod
metadata:
  name: scalable-cr
  namespace: default
spec:
  selector: "app=scalable-cr" 1
  replicas: 1

```

- 1** Specify the label type to apply to managed pods. This is the field that the **labelSelectorPath** references in the custom resource definition object.

Example VPA object

```

apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: scalable-cr
  namespace: default
spec:

```

```
targetRef:
  apiVersion: testing.openshift.io/v1
  kind: ScalablePod
  name: scalable-cr
updatePolicy:
  updateMode: "Auto"
```

2.5.6. Uninstalling the Vertical Pod Autoscaler Operator

You can remove the Vertical Pod Autoscaler Operator (VPA) from your OpenShift Container Platform cluster. After uninstalling, the resource requests for the pods that are already modified by an existing VPA custom resource (CR) do not change. The resources defined in the workload object, not the previous recommendations made by the VPA, are allocated to any new pods.



NOTE

You can remove a specific VPA CR by using the **oc delete vpa <vpa-name>** command. The same actions apply for resource requests as uninstalling the vertical pod autoscaler.

After removing the VPA, it is recommended that you remove the other components associated with the Operator to avoid potential issues.

Prerequisites

- You installed the VPA.

Procedure

1. In the OpenShift Container Platform web console, click **Ecosystem** → **Installed Operators**.
2. Switch to the **openshift-vertical-pod-autoscaler** project.
3. For the **VerticalPodAutoscaler** Operator, click the Options menu  and select **Uninstall Operator**.
4. Optional: To remove all operands associated with the Operator, in the dialog box, select **Delete all operand instances for this operator** checkbox.
5. Click **Uninstall**.
6. Optional: Use the OpenShift CLI to remove the VPA components:
 - a. Delete the VPA namespace:

```
$ oc delete namespace openshift-vertical-pod-autoscaler
```

- b. Delete the VPA custom resource definition (CRD) objects:

```
$ oc delete crd verticalpodautoscalercheckpoints.autoscaling.k8s.io
```

```
$ oc delete crd verticalpodautoscalercontrollers.autoscaling.openshift.io
```

```
$ oc delete crd verticalpodautoscalers.autoscaling.k8s.io
```

Deleting the CRDs removes the associated roles, cluster roles, and role bindings.



NOTE

This action removes from the cluster all user-created VPA CRs. If you re-install the VPA, you must create these objects again.

- c. Delete the **MutatingWebhookConfiguration** object by running the following command:

```
$ oc delete MutatingWebhookConfiguration vpa-webhook-config
```

- d. Delete the VPA Operator:

```
$ oc delete operator/vertical-pod-autoscaler.openshift-vertical-pod-autoscaler
```

2.6. ADJUST POD RESOURCE LEVELS WITHOUT POD DISRUPTION

You can change the CPU or memory resource requests and limits assigned to a container without re-creating or restarting the pod by using *in-place pod resizing*.

2.6.1. About in-place pod resizing

In-place pod resizing allows you to change the CPU and memory resources for containers within a running pod without application disruption. The standard methods for changing pod CPU and memory resources cause the pod to be re-created, potentially causing disruption. In-place pod resizing allows you to scale pod resources up or down without suffering the downtime or state loss associated with a pod restart.

When using in-place pod resizing to change CPU or memory resources, you can control whether a pod is restarted by configuring a resize policy in the pod specification. The following example resize policy requires a pod restart upon changing the memory resources, but prevents a restart for CPU resource changes.

Example resource policy

```
apiVersion: v1
kind: Pod
metadata:
  name: resize-demo
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: pause
# ...
    resizePolicy: 1
    - resourceName: cpu
```



```
restartPolicy: NotRequired
- resourceName: memory
  restartPolicy: RestartContainer
```

- 1 Specifies a resize policy.



NOTE

Memory limits cannot be decreased unless the resize policy for **memory** is **RestartContainer**.

You cannot add or modify a resize policy to an existing pod, but you can add or edit the policy in the pod's owner object, such as a deployment, if the pod has an owner object.

Using in-place pod resizing requires that you use the **--subresource resize** flag when editing a pod in the OpenShift CLI (**oc**), as shown in the following examples:

Example commands

```
$ oc edit pod <pod_name> --subresource resize
```

```
$ oc apply -f <file_name>.yaml --subresource resize
```

```
$ oc patch pod <pod_name> --subresource resize --patch \
  '{"spec":{"containers":[{"name":"pause", "resources":{"requests":{"cpu":"800m"}, "limits":
  {"cpu":"800m"}}}]}}'
```

Because you need to use the **--subresource resize** flag with a resize policy, you cannot edit the pod resources in the OpenShift Container Platform web console.

If the resize policy is **NotRequired** and you change the request or limits, the pod is not restarted.

```
$ oc get pods
```

Example output

NAME	READY	STATUS	RESTARTS	AGE
resize-pod	1/1	Running	0	5s

If the resize policy is **RestartContainer** and you change the request or limits, the pod is restarted.

```
$ oc get pods
```

Example output

NAME	READY	STATUS	RESTARTS	AGE
resize-pod	1/1	Running	1 (5s ago)	5s

After making the resource changes, the pod status conditions indicate the state of a resize request by using the following messages:

- **PodResizeInProgress:** The kubelet is able to allocate the requested resources and the change is being applied.
- **PodResizePending:** The kubelet cannot immediately make the change for one of the following reasons:
 - **Infeasible:** The requested resize cannot be executed on the current node. For example, requesting more resources than the node has available would result in an **Infeasible** condition.
 - **Deferred:** The requested resize is currently not possible, but might become possible at a later time. For example, if another pod is removed from the node, the requested resources might become available. The kubelet retries the resize when conditions on the node change.
- **Error:** The kubelet is experiencing an error during the resource allocation and reports the reason for the error in the message field.

Example status for an infeasible change

```
apiVersion: v1
kind: Pod
metadata:
  name: resize-demo
# ...
status:
  conditions:
  - lastProbeTime: "2025-09-03T15:00:50Z"
    lastTransitionTime: "2025-09-03T15:00:50Z"
    message: 'Node didn"t have enough capacity: cpu, requested: 1000000, capacity:
      3500'
    reason: Infeasible
    status: "True"
    type: PodResizePending
```

Note the following limitations:

- In-place pod resizing is not supported for non-restartable init containers and ephemeral containers.
- In-place pod resizing is not allowed if the changes violate other pod mutability constraints, such as the pod QoS class.
- Pods managed by a static **cpuManagerPolicy** or **memoryManagerPolicy** parameter cannot be resized with in-place pod resizing.
- Pods utilizing swap memory must use the **RestartContainer** policy for memory requests with in-place pod resizing.

2.6.2. Configuring in-place pod resizing

In-place pod resizing requires that you add a resize policy to a pod specification.

You cannot add or modify a resize policy in an existing pod, but you can add or edit the policy in the pod's owner object, such as a deployment, if the pod has an owner object.

Procedure

1. Create a pod spec with a resize policy or add a resize policy to the owner object of an existing pod:
 - a. Create a YAML file similar to the following example:

```
apiVersion: v1
kind: Pod
metadata:
  name: resize-pod
spec:
  # ...
  containers:
  - name: pause
    resizePolicy: 1
    - resourceName: cpu
      restartPolicy: NotRequired
    - resourceName: memory
      restartPolicy: RestartContainer
  # ...
```

- 1** Specifies a resize policy. For CPU and/or memory resources specify one of the following values:
- **NotRequired:** Apply any resource changes without restarting the pod. This is the default when using a resize policy.
 - **RestartContainer:** Apply any resource changes and restart the pod.

- b. Create the object by running a command similar to the following:

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

Verification

- Check that the resize policy is applied by modifying the CPU or memory requests or limits by running a command similar to the following. You must include the **--subresource resize** flag. If the pod has a owner object, such as a deployment, you must edit the owner object.

```
$ oc edit pod <pod_name> --subresource resize
```

If the policy is applied, the pod responds as expected.

```
$ oc get pods
```

If the resize policy is **NotRequired**, the pod is not restarted.

Example output

NAME	READY	STATUS	RESTARTS	AGE
resize-pod	1/1	Running	0	5s

If the resize policy is **RestartContainer**, the pod is restarted.

Example output

NAME	READY	STATUS	RESTARTS	AGE
resize-pod	1/1	Running	1 (5s ago)	5s

2.6.3. Additional resources

- [Understanding resource requests and limits](#)

2.7. PROVIDING SENSITIVE DATA TO PODS BY USING SECRETS

Some applications need sensitive information, such as passwords and user names, that you do not want developers to have.

As an administrator, you can use **Secret** objects to provide this information without exposing that information in clear text.

2.7.1. Understanding secrets

The **Secret** object type provides a mechanism to hold sensitive information such as passwords, OpenShift Container Platform client configuration files, private source repository credentials, and so on. Secrets decouple sensitive content from the pods. You can mount secrets into containers using a volume plugin or the system can use secrets to perform actions on behalf of a pod.

Key properties include:

- Secret data can be referenced independently from its definition.
- Secret data volumes are backed by temporary file-storage facilities (tmpfs) and never come to rest on a node.
- Secret data can be shared within a namespace.

YAML Secret object definition

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
  namespace: my-namespace
type: Opaque ①
data: ②
  username: <username> ③
  password: <password>
stringData: ④
  hostname: myapp.mydomain.com ⑤
```

- ① Indicates the structure of the secret's key names and values.
- ② The allowable format for the keys in the **data** field must meet the guidelines in the **DNS_SUBDOMAIN** value in [the Kubernetes identifiers glossary](#).

- 3 The value associated with keys in the **data** map must be base64 encoded.
- 4 Entries in the **stringData** map are converted to base64 and the entry will then be moved to the **data** map automatically. This field is write-only; the value will only be returned via the **data** field.
- 5 The value associated with keys in the **stringData** map is made up of plain text strings.

You must create a secret before creating the pods that depend on that secret.

When creating secrets:

- Create a secret object with secret data.
- Update the pod's service account to allow the reference to the secret.
- Create a pod, which consumes the secret as an environment variable or as a file (using a **secret** volume).

2.7.1.1. Types of secrets

The value in the **type** field indicates the structure of the secret's key names and values. The type can be used to enforce the presence of user names and keys in the secret object. If you do not want validation, use the **opaque** type, which is the default.

Specify one of the following types to trigger minimal server-side validation to ensure the presence of specific key names in the secret data:

- **kubernetes.io/basic-auth**: Use with Basic authentication
- **kubernetes.io/dockercfg**: Use as an image pull secret
- **kubernetes.io/dockerconfigjson**: Use as an image pull secret
- **kubernetes.io/service-account-token**: Use to obtain a legacy service account API token
- **kubernetes.io/ssh-auth**: Use with SSH key authentication
- **kubernetes.io/tls**: Use with TLS certificate authorities

Specify **type: Opaque** if you do not want validation, which means the secret does not claim to conform to any convention for key names or values. An *opaque* secret, allows for unstructured **key:value** pairs that can contain arbitrary values.



NOTE

You can specify other arbitrary types, such as **example.com/my-secret-type**. These types are not enforced server-side, but indicate that the creator of the secret intended to conform to the key/value requirements of that type.

For examples of creating different types of secrets, see *Understanding how to create secrets*.

2.7.1.2. Secret data keys

Secret keys must be in a DNS subdomain.

2.7.1.3. Automatically generated image pull secrets

By default, OpenShift Container Platform creates an image pull secret for each service account.



NOTE

Prior to OpenShift Container Platform 4.16, a long-lived service account API token secret was also generated for each service account that was created. Starting with OpenShift Container Platform 4.16, this service account API token secret is no longer created.

After upgrading to 4.20, any existing long-lived service account API token secrets are not deleted and will continue to function. For information about detecting long-lived API tokens that are in use in your cluster or deleting them if they are not needed, see the Red Hat Knowledgebase article [Long-lived service account API tokens in OpenShift Container Platform](#).

This image pull secret is necessary to integrate the OpenShift image registry into the cluster's user authentication and authorization system.

However, if you do not enable the **ImageRegistry** capability or if you disable the integrated OpenShift image registry in the Cluster Image Registry Operator's configuration, an image pull secret is not generated for each service account.

When the integrated OpenShift image registry is disabled on a cluster that previously had it enabled, the previously generated image pull secrets are deleted automatically.

2.7.2. Understanding how to create secrets

As an administrator you must create a secret before developers can create the pods that depend on that secret.

When creating secrets:

1. Create a secret object that contains the data you want to keep secret. The specific data required for each secret type is described in the following sections.

Example YAML object that creates an opaque secret

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
type: Opaque ①
data: ②
  username: <username>
  password: <password>
stringData: ③
  hostname: myapp.mydomain.com
secret.properties: |
  property1=valueA
  property2=valueB
```

- ① Specifies the type of secret.

- 2 Specifies encoded string and data.
- 3 Specifies decoded string and data.

Use either the **data** or **stringdata** fields, not both.

2. Update the pod's service account to reference the secret:

YAML of a service account that uses a secret

```
apiVersion: v1
kind: ServiceAccount
...
secrets:
- name: test-secret
```

3. Create a pod, which consumes the secret as an environment variable or as a file (using a **secret** volume):

YAML of a pod populating files in a volume with secret data

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: secret-test-container
    image: busybox
    command: [ "/bin/sh", "-c", "cat /etc/secret-volume/*" ]
    volumeMounts: 1
      - name: secret-volume
        mountPath: /etc/secret-volume 2
        readOnly: true 3
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  volumes:
  - name: secret-volume
    secret:
      secretName: test-secret 4
  restartPolicy: Never
```

- 1 Add a **volumeMounts** field to each container that needs the secret.
- 2 Specifies an unused directory name where you would like the secret to appear. Each key in the secret data map becomes the filename under **mountPath**.

- 3 Set to **true**. If true, this instructs the driver to provide a read-only volume.
- 4 Specifies the name of the secret.

YAML of a pod populating environment variables with secret data

```

apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "export" ]
      env:
        - name: TEST_SECRET_USERNAME_ENV_VAR
          valueFrom:
            secretKeyRef: 1
              name: test-secret
              key: username
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
      restartPolicy: Never

```

- 1 Specifies the environment variable that consumes the secret key.

YAML of a build config populating environment variables with secret data

```

apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
      env:
        - name: TEST_SECRET_USERNAME_ENV_VAR
          valueFrom:
            secretKeyRef: 1
              name: test-secret
              key: username
      from:
        kind: ImageStreamTag
        namespace: openshift
        name: 'cli:latest'

```


- 1 Specifies the environment variable that consumes the secret key.

2.7.2.1. Secret creation restrictions

To use a secret, a pod needs to reference the secret. A secret can be used with a pod in three ways:

- To populate environment variables for containers.
- As files in a volume mounted on one or more of its containers.
- By kubelet when pulling images for the pod.

Volume type secrets write data into the container as a file using the volume mechanism. Image pull secrets use service accounts for the automatic injection of the secret into all pods in a namespace.

When a template contains a secret definition, the only way for the template to use the provided secret is to ensure that the secret volume sources are validated and that the specified object reference actually points to a **Secret** object. Therefore, a secret needs to be created before any pods that depend on it. The most effective way to ensure this is to have it get injected automatically through the use of a service account.

Secret API objects reside in a namespace. They can only be referenced by pods in that same namespace.

Individual secrets are limited to 1MB in size. This is to discourage the creation of large secrets that could exhaust apiserver and kubelet memory. However, creation of a number of smaller secrets could also exhaust memory.

2.7.2.2. Creating an opaque secret

As an administrator, you can create an opaque secret, which allows you to store unstructured **key:value** pairs that can contain arbitrary values.

Procedure

1. Create a **Secret** object in a YAML file.

For example:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque 1
data:
  username: <username>
  password: <password>
```

- 1 Specifies an opaque secret.

2. Use the following command to create a **Secret** object:

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

3. To use the secret in a pod:

- a. Update the pod's service account to reference the secret, as shown in the "Understanding how to create secrets" section.
- b. Create the pod, which consumes the secret as an environment variable or as a file (using a **secret** volume), as shown in the "Understanding how to create secrets" section.

Additional resources

- [Understanding how to create secrets](#)

2.7.2.3. Creating a legacy service account token secret

As an administrator, you can create a legacy service account token secret, which allows you to distribute a service account token to applications that must authenticate to the API.



WARNING

It is recommended to obtain bound service account tokens using the TokenRequest API instead of using legacy service account token secrets. You should create a service account token secret only if you cannot use the TokenRequest API and if the security exposure of a nonexpiring token in a readable API object is acceptable to you.

Bound service account tokens are more secure than service account token secrets for the following reasons:

- Bound service account tokens have a bounded lifetime.
- Bound service account tokens contain audiences.
- Bound service account tokens can be bound to pods or secrets and the bound tokens are invalidated when the bound object is removed.

Workloads are automatically injected with a projected volume to obtain a bound service account token. If your workload needs an additional service account token, add an additional projected volume in your workload manifest.

For more information, see "Configuring bound service account tokens using volume projection".

Procedure

1. Create a **Secret** object in a YAML file:

Example Secret object

```
apiVersion: v1
kind: Secret
metadata:
```

```
name: secret-sa-sample
annotations:
  kubernetes.io/service-account.name: "sa-name" 1
type: kubernetes.io/service-account-token 2
```

1 Specifies an existing service account name. If you are creating both the **ServiceAccount** and the **Secret** objects, create the **ServiceAccount** object first.

2 Specifies a service account token secret.

2. Use the following command to create the **Secret** object:

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

3. To use the secret in a pod:

- a. Update the pod's service account to reference the secret, as shown in the "Understanding how to create secrets" section.
- b. Create the pod, which consumes the secret as an environment variable or as a file (using a **secret** volume), as shown in the "Understanding how to create secrets" section.

Additional resources

- [Understanding how to create secrets](#)
- [Configuring bound service account tokens using volume projection](#)
- [Understanding and creating service accounts](#)

2.7.2.4. Creating a basic authentication secret

As an administrator, you can create a basic authentication secret, which allows you to store the credentials needed for basic authentication. When using this secret type, the **data** parameter of the **Secret** object must contain the following keys encoded in the base64 format:

- **username**: the user name for authentication
- **password**: the password or token for authentication



NOTE

You can use the **stringData** parameter to use clear text content.

Procedure

1. Create a **Secret** object in a YAML file:

Example secret object

```
apiVersion: v1
kind: Secret
metadata:
```

```

name: secret-basic-auth
type: kubernetes.io/basic-auth 1
data:
  stringData: 2
    username: admin
    password: <password>

```

- 1** Specifies a basic authentication secret.
- 2** Specifies the basic authentication values to use.

2. Use the following command to create the **Secret** object:

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

3. To use the secret in a pod:
 - a. Update the pod's service account to reference the secret, as shown in the "Understanding how to create secrets" section.
 - b. Create the pod, which consumes the secret as an environment variable or as a file (using a **secret** volume), as shown in the "Understanding how to create secrets" section.

Additional resources

- [Understanding how to create secrets](#)

2.7.2.5. Creating an SSH authentication secret

As an administrator, you can create an SSH authentication secret, which allows you to store data used for SSH authentication. When using this secret type, the **data** parameter of the **Secret** object must contain the SSH credential to use.

Procedure

1. Create a **Secret** object in a YAML file on a control plane node:

Example secret object

```

apiVersion: v1
kind: Secret
metadata:
  name: secret-ssh-auth
type: kubernetes.io/ssh-auth 1
data:
  ssh-privatekey: | 2
    MIIEpQIBAAKCAQEAulqb/Y ...

```

- 1** Specifies an SSH authentication secret.
- 2** Specifies the SSH key/value pair as the SSH credentials to use.

2. Use the following command to create the **Secret** object:

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

3. To use the secret in a pod:

- a. Update the pod's service account to reference the secret, as shown in the "Understanding how to create secrets" section.
- b. Create the pod, which consumes the secret as an environment variable or as a file (using a **secret** volume), as shown in the "Understanding how to create secrets" section.

Additional resources

- [Understanding how to create secrets](#)

2.7.2.6. Creating a Docker configuration secret

As an administrator, you can create a Docker configuration secret, which allows you to store the credentials for accessing a container image registry.

- **kubernetes.io/dockercfg**. Use this secret type to store your local Docker configuration file. The **data** parameter of the **secret** object must contain the contents of a **.dockercfg** file encoded in the base64 format.
- **kubernetes.io/dockerconfigjson**. Use this secret type to store your local Docker configuration JSON file. The **data** parameter of the **secret** object must contain the contents of a **.docker/config.json** file encoded in the base64 format.

Procedure

1. Create a **Secret** object in a YAML file.

Example Docker configuration secret object

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-docker-cfg
  namespace: my-project
type: kubernetes.io/dockerconfig 1
data:

.dockerconfig:bm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2cgYXV
0aCBzXlZCg== 2
```

1 Specifies that the secret is using a Docker configuration file.

2 The output of a base64-encoded Docker configuration file

Example Docker configuration JSON secret object

```
apiVersion: v1
```

```

kind: Secret
metadata:
  name: secret-docker-json
  namespace: my-project
type: kubernetes.io/dockerconfig 1
data:

.dockerconfigjson:bm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2cg
YXV0aCBrZXlzcG== 2

```

- 1** Specifies that the secret is using a Docker configuration JSONfile.
- 2** The output of a base64-encoded Docker configuration JSON file

2. Use the following command to create the **Secret** object

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

3. To use the secret in a pod:
 - a. Update the pod's service account to reference the secret, as shown in the "Understanding how to create secrets" section.
 - b. Create the pod, which consumes the secret as an environment variable or as a file (using a **secret** volume), as shown in the "Understanding how to create secrets" section.

Additional resources

- [Understanding how to create secrets](#)

2.7.2.7. Creating a secret using the web console

You can create secrets using the web console.

Procedure

1. Navigate to **Workloads → Secrets**.
2. Click **Create → From YAML**.
 - a. Edit the YAML manually to your specifications, or drag and drop a file into the YAML editor.
For example:

```

apiVersion: v1
kind: Secret
metadata:
  name: example
  namespace: <namespace>
type: Opaque 1
data:
  username: <base64 encoded username>
  password: <base64 encoded password>
stringData: 2
  hostname: myapp.mydomain.com

```

- 1 This example specifies an opaque secret; however, you may see other secret types such as service account token secret, basic authentication secret, SSH authentication secret, or a secret that uses Docker configuration.
- 2 Entries in the **stringData** map are converted to base64 and the entry will then be moved to the **data** map automatically. This field is write-only; the value will only be returned via the **data** field.

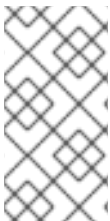
3. Click **Create**.
4. Click **Add Secret to workload**
 - a. From the drop-down menu, select the workload to add.
 - b. Click **Save**.

2.7.3. Understanding how to update secrets

When you modify the value of a secret, the value (used by an already running pod) will not dynamically change. To change a secret, you must delete the original pod and create a new pod (perhaps with an identical PodSpec).

Updating a secret follows the same workflow as deploying a new Container image. You can use the **kubectl rolling-update** command.

The **resourceVersion** value in a secret is not specified when it is referenced. Therefore, if a secret is updated at the same time as pods are starting, the version of the secret that is used for the pod is not defined.



NOTE

Currently, it is not possible to check the resource version of a secret object that was used when a pod was created. It is planned that pods will report this information, so that a controller could restart ones using an old **resourceVersion**. In the interim, do not update the data of existing secrets, but create new ones with distinct names.

2.7.4. Creating and using secrets

As an administrator, you can create a service account token secret. This allows you to distribute a service account token to applications that must authenticate to the API.

Procedure

1. Create a service account in your namespace by running the following command:

```
$ oc create sa <service_account_name> -n <your_namespace>
```

2. Save the following YAML example to a file named **service-account-token-secret.yaml**. The example includes a **Secret** object configuration that you can use to generate a service account token:

```
apiVersion: v1
kind: Secret
```

```

metadata:
  name: <secret_name> ❶
  annotations:
    kubernetes.io/service-account.name: "sa-name" ❷
type: kubernetes.io/service-account-token ❸

```

- ❶ Replace **<secret_name>** with the name of your service token secret.
- ❷ Specifies an existing service account name. If you are creating both the **ServiceAccount** and the **Secret** objects, create the **ServiceAccount** object first.
- ❸ Specifies a service account token secret type.

3. Generate the service account token by applying the file:

```
$ oc apply -f service-account-token-secret.yaml
```

4. Get the service account token from the secret by running the following command:

```
$ oc get secret <sa_token_secret> -o jsonpath='{.data.token}' | base64 --decode ❶
```

Example output

```

ayJhbGciOiJSUzI1NiIsImtpZCI6IklOb2dtck1qZ3hCSWpoNnh5YnZhSE9QMkk3YnRZMVZoclFf
QTZfRFP1YlUiifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50liwia3ViZXJuZXRlcy5pby9zZXJ2aWNIYWNjb3VudC9uYW1lc3BhY2UiOiJkZWZhdWx0liwia3ViZXJuZXRlcy5pby9zZXJ2aWNIYWNjb3VudC9zZWNyZXQubmFtZSI6ImJ1aWxkZXItZG9rZW4tdHZrbnliLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoieVpGRlcilmt1YmVybmV0ZXMuaW8vc2VydmljZWZjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6IjNmZGU2MGZmLTA1NGYtNDkyZi04YzhjLTNlcnZjE0NDk3MmFmNylsInN1YiI6InN5c3RlbnVpZzZXJ2aWNIYWNjb3VudDpkZWZhdWx0OmJ1aWxkZXIifQ.OmqFTDuMHC_IYvvEUrjr1x453hIEEHYcxS9VK
SzmRkP1SiVZWPNPkTWIfNRp6bIUZD3U6aN3N7dMSN0eI5hu36xPgpKTdvuckKLTcneIMx6c
xOdAbrcw1mCmOCINscwjS1KO1kzMtYnnq8rXHIMJELsNIhnRyyIXRTtNBsy4t64T3283s3SLsancyx0gy0ujx-Ch3uKAKdZi5iT-l8jnnQ-ds5THDs2h65RjhggIQEmSxpHrLGZFmyHAQI-
_SjvmHZPXEc482x3SkaQHNLqpmrpJorNqh1M8ZHKzlujhZgVooMvJmWPXTb2vnvi3DGn2Xl-hZxl1yD2yGH1RBpYUHA

```

- ❶ Replace **<sa_token_secret>** with the name of your service token secret.

5. Use your service account token to authenticate with the API of your cluster:

```
$ curl -X GET <openshift_cluster_api> --header "Authorization: Bearer <token>" ❶ ❷
```

- ❶ Replace **<openshift_cluster_api>** with the OpenShift cluster API.
- ❷ Replace **<token>** with the service account token that is output in the preceding command.

2.7.5. About using signed certificates with secrets

To secure communication to your service, you can configure OpenShift Container Platform to generate a signed serving certificate/key pair that you can add into a secret in a project.

A *service serving certificate secret* is intended to support complex middleware applications that need out-of-the-box certificates. It has the same settings as the server certificates generated by the administrator tooling for nodes and masters.

Service Pod spec configured for a service serving certificates secret.

```
apiVersion: v1
kind: Service
metadata:
  name: registry
  annotations:
    service.beta.openshift.io/serving-cert-secret-name: registry-cert 1
# ...
```

- 1** Specify the name for the certificate

Other pods can trust cluster-created certificates (which are only signed for internal DNS names), by using the CA bundle in the `/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt` file that is automatically mounted in their pod.

The signature algorithm for this feature is **x509.SHA256WithRSA**. To manually rotate, delete the generated secret. A new certificate is created.

2.7.5.1. Generating signed certificates for use with secrets

To use a signed serving certificate/key pair with a pod, create or edit the service to add the **service.beta.openshift.io/serving-cert-secret-name** annotation, then add the secret to the pod.

Procedure

To create a *service serving certificate secret*:

1. Edit the **Pod** spec for your service.
2. Add the **service.beta.openshift.io/serving-cert-secret-name** annotation with the name you want to use for your secret.

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
  annotations:
    service.beta.openshift.io/serving-cert-secret-name: my-cert 1
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

- 1 The certificate and key are in PEM format, stored in **tls.crt** and **tls.key** respectively.

3. Create the service:

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

4. View the secret to make sure it was created:

- a. View a list of all secrets:

```
$ oc get secrets
```

Example output

NAME	TYPE	DATA	AGE
my-cert	kubernetes.io/tls	2	9m

- b. View details on your secret:

```
$ oc describe secret my-cert
```

Example output

```
Name:      my-cert
Namespace: openshift-console
Labels:    <none>
Annotations: service.beta.openshift.io/expiry: 2023-03-08T23:22:40Z
             service.beta.openshift.io/originating-service-name: my-service
             service.beta.openshift.io/originating-service-uid: 640f0ec3-afc2-4380-bf31-a8c784846a11
             service.beta.openshift.io/expiry: 2023-03-08T23:22:40Z

Type: kubernetes.io/tls

Data
====
tls.key: 1679 bytes
tls.crt: 2595 bytes
```

5. Edit your **Pod** spec with that secret.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-service-pod
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
  - name: mypod
    image: redis
```

```

volumeMounts:
- name: my-container
  mountPath: "/etc/my-path"
securityContext:
  allowPrivilegeEscalation: false
  capabilities:
    drop: [ALL]
volumes:
- name: my-volume
  secret:
    secretName: my-cert
    items:
    - key: username
      path: my-group/my-username
      mode: 511

```

When it is available, your pod will run. The certificate will be good for the internal service DNS name, **<service.name>.<service.namespace>.svc**.

The certificate/key pair is automatically replaced when it gets close to expiration. View the expiration date in the **service.beta.openshift.io/expiry** annotation on the secret, which is in RFC3339 format.



NOTE

In most cases, the service DNS name **<service.name>.<service.namespace>.svc** is not externally routable. The primary use of **<service.name>.<service.namespace>.svc** is for intracluster or intraservice communication, and with re-encrypt routes.

2.7.6. Troubleshooting secrets

If a service certificate generation fails with (service's **service.beta.openshift.io/serving-cert-generation-error** annotation contains):

```
secret/ssl-key references serviceUID 62ad25ca-d703-11e6-9d6f-0e9c0057b608, which does not match 77b6dd80-d716-11e6-9d6f-0e9c0057b60
```

The service that generated the certificate no longer exists, or has a different **serviceUID**. You must force certificates regeneration by removing the old secret, and clearing the following annotations on the service **service.beta.openshift.io/serving-cert-generation-error**, **service.beta.openshift.io/serving-cert-generation-error-num**:

1. Delete the secret:

```
$ oc delete secret <secret_name>
```

2. Clear the annotations:

```
$ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-
```

```
$ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-num-
```

**NOTE**

The command removing annotation has a - after the annotation name to be removed.

2.8. PROVIDING SENSITIVE DATA TO PODS BY USING AN EXTERNAL SECRETS STORE

Some applications need sensitive information, such as passwords and user names, that you do not want developers to have.

As an alternative to using Kubernetes **Secret** objects to provide sensitive information, you can use an external secrets store to store the sensitive information. You can use the Secrets Store CSI Driver Operator to integrate with an external secrets store and mount the secret content as a pod volume.

2.8.1. About the Secrets Store CSI Driver Operator

Kubernetes secrets are stored with Base64 encoding. etcd provides encryption at rest for these secrets, but when secrets are retrieved, they are decrypted and presented to the user. If role-based access control is not configured properly on your cluster, anyone with API or etcd access can retrieve or modify a secret. Additionally, anyone who is authorized to create a pod in a namespace can use that access to read any secret in that namespace.

To store and manage your secrets securely, you can configure the OpenShift Container Platform Secrets Store Container Storage Interface (CSI) Driver Operator to mount secrets from an external secret management system, such as Azure Key Vault, by using a provider plugin. Applications can then use the secret, but the secret does not persist on the system after the application pod is destroyed.

The Secrets Store CSI Driver Operator, **secrets-store.csi.k8s.io**, enables OpenShift Container Platform to mount multiple secrets, keys, and certificates stored in enterprise-grade external secrets stores into pods as a volume. The Secrets Store CSI Driver Operator communicates with the provider using gRPC to fetch the mount contents from the specified external secrets store. After the volume is attached, the data in it is mounted into the container's file system. Secrets store volumes are mounted in-line.

2.8.1.1. Secrets store providers

The Secrets Store CSI Driver Operator has been tested with the following secrets store providers:

- AWS Secrets Manager
- AWS Systems Manager Parameter Store
- Azure Key Vault
- Google Secret Manager
- HashiCorp Vault



NOTE

Red Hat does not test all factors associated with third-party secrets store provider functionality. For more information about third-party support, see the [Red Hat third-party support policy](#).

2.8.1.2. Automatic rotation

The Secrets Store CSI driver periodically rotates the content in the mounted volume with the content from the external secrets store. If a secret is updated in the external secrets store, the secret will be updated in the mounted volume. The Secrets Store CSI Driver Operator polls for updates every 2 minutes.

If you enabled synchronization of mounted content as Kubernetes secrets, the Kubernetes secrets are also rotated.

Applications consuming the secret data must watch for updates to the secrets.

2.8.2. Installing the Secrets Store CSI driver

Prerequisites

- Access to the OpenShift Container Platform web console.
- Administrator access to the cluster.

Procedure

To install the Secrets Store CSI driver:

1. Install the Secrets Store CSI Driver Operator:
 - a. Log in to the web console.
 - b. Click **Ecosystem** → **Software Catalog**.
 - c. Locate the Secrets Store CSI Driver Operator by typing "Secrets Store CSI" in the filter box.
 - d. Click the **Secrets Store CSI Driver Operator** button.
 - e. On the **Secrets Store CSI Driver Operator** page, click **Install**.
 - f. On the **Install Operator** page, ensure that:
 - **All namespaces on the cluster (default)** is selected.
 - **Installed Namespace** is set to **openshift-cluster-csi-drivers**.
 - g. Click **Install**.
After the installation finishes, the Secrets Store CSI Driver Operator is listed in the **Installed Operators** section of the web console.
2. Create the **ClusterCSIDriver** instance for the driver (**secrets-store.csi.k8s.io**):
 - a. Click **Administration** → **CustomResourceDefinitions** → **ClusterCSIDriver**.
 - b. On the **Instances** tab, click **Create ClusterCSIDriver**.

Use the following YAML file:

```
apiVersion: operator.openshift.io/v1
kind: ClusterCSIDriver
metadata:
  name: secrets-store.csi.k8s.io
spec:
  managementState: Managed
```

c. Click **Create**.

2.8.3. Mounting secrets from an external secrets store to a CSI volume

After installing the Secrets Store CSI Driver Operator, you can mount secrets from one of the following external secrets stores to a CSI volume:

- [AWS Secrets Manager](#)
- [AWS Systems Manager Parameter Store](#)
- [Azure Key Vault](#)
- [Google Secret Manager](#)
- [HashiCorp Vault](#)

2.8.3.1. Mounting secrets from AWS Secrets Manager

You can use the Secrets Store CSI Driver Operator to mount secrets from AWS Secrets Manager external secrets store to a Container Storage Interface (CSI) volume in OpenShift Container Platform.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the **jq** tool.
- You have extracted and prepared the **ccocctl** utility.
- You have installed the cluster on Amazon Web Services (AWS) and the cluster uses AWS Security Token Service (STS).
- You have installed the Secrets Store CSI Driver Operator. For more information, see "Installing the Secrets Store CSI driver".
- You have configured AWS Secrets Manager to store the required secrets.

Procedure

1. Install the AWS Secrets Manager provider:
 - a. Create a YAML file by using the following example configuration:



IMPORTANT

The AWS Secrets Manager provider for the Secrets Store CSI driver is an upstream provider.

This configuration is modified from the configuration provided in the upstream [AWS documentation](#) so that it works properly with OpenShift Container Platform. Changes to this configuration might impact functionality.

Example `aws-provider.yaml` file

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: csi-secrets-store-provider-aws
  namespace: openshift-cluster-csi-drivers
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: csi-secrets-store-provider-aws-cluster-role
rules:
- apiGroups: [""]
  resources: ["serviceaccounts/token"]
  verbs: ["create"]
- apiGroups: [""]
  resources: ["serviceaccounts"]
  verbs: ["get"]
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get"]
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: csi-secrets-store-provider-aws-cluster-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: csi-secrets-store-provider-aws-cluster-role
subjects:
- kind: ServiceAccount
  name: csi-secrets-store-provider-aws
  namespace: openshift-cluster-csi-drivers
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  namespace: openshift-cluster-csi-drivers
  name: csi-secrets-store-provider-aws
  labels:
    app: csi-secrets-store-provider-aws
spec:
```

```

updateStrategy:
  type: RollingUpdate
selector:
  matchLabels:
    app: csi-secrets-store-provider-aws
template:
  metadata:
    labels:
      app: csi-secrets-store-provider-aws
  spec:
    serviceAccountName: csi-secrets-store-provider-aws
    hostNetwork: false
    containers:
      - name: provider-aws-installer
        image: public.ecr.aws/aws-secrets-manager/secrets-store-csi-driver-provider-aws:1.0.r2-50-g5b4aca1-2023.06.09.21.19
        imagePullPolicy: Always
        args:
          - --provider-volume=/etc/kubernetes/secrets-store-csi-providers
        resources:
          requests:
            cpu: 50m
            memory: 100Mi
          limits:
            cpu: 50m
            memory: 100Mi
        securityContext:
          privileged: true
        volumeMounts:
          - mountPath: "/etc/kubernetes/secrets-store-csi-providers"
            name: providervol
          - name: mountpoint-dir
            mountPath: /var/lib/kubelet/pods
            mountPropagation: HostToContainer
    tolerations:
      - operator: Exists
    volumes:
      - name: providervol
        hostPath:
          path: "/etc/kubernetes/secrets-store-csi-providers"
      - name: mountpoint-dir
        hostPath:
          path: /var/lib/kubelet/pods
          type: DirectoryOrCreate
    nodeSelector:
      kubernetes.io/os: linux

```

- b. Grant privileged access to the **csi-secrets-store-provider-aws** service account by running the following command:

```
$ oc adm policy add-scc-to-user privileged -z csi-secrets-store-provider-aws -n openshift-cluster-csi-drivers
```

- c. Create the provider resources by running the following command:


```
$ oc apply -f aws-provider.yaml
```

2. Grant the read permission to the service account for the AWS secret object:

a. Create a directory to contain the credentials request by running the following command:

```
$ mkdir <aws_creds_directory_name>
```

b. Create a YAML file that defines the **CredentialsRequest** resource configuration. See the following example configuration:

```
apiVersion: cloudcredential.openshift.io/v1
kind: CredentialsRequest
metadata:
  name: aws-creds-request
  namespace: openshift-cloud-credential-operator
spec:
  providerSpec:
    apiVersion: cloudcredential.openshift.io/v1
    kind: AWSProviderSpec
    statementEntries:
      - action:
        - "secretsmanager:GetSecretValue"
        - "secretsmanager:DescribeSecret"
        effect: Allow
        resource: "arn:*.secretsmanager:*.*:secret:testSecret-?????"
  secretRef:
    name: aws-creds
    namespace: my-namespace
  serviceAccountNames:
    - <service_account_name>
```

c. Retrieve the OpenID Connect (OIDC) provider by running the following command:

```
$ oc get --raw=/well-known/openid-configuration | jq -r '.issuer'
```

Example output

```
https://<oidc_provider_name>
```

Copy the OIDC provider name **<oidc_provider_name>** from the output to use in the next step.

d. Use the **ccoctl** tool to process the credentials request by running the following command:

```
$ ccoctl aws create-iam-roles \
  --name my-role --region=<aws_region> \
  --credentials-requests-dir=<aws_creds_dir_name> \
  --identity-provider-arn arn:aws:iam::<aws_account_id>:oidc-
  provider/<oidc_provider_name> --output-dir=<output_dir_name>
```

Example output

```

2023/05/15 18:10:34 Role arn:aws:iam::<aws_account_id>:role/my-role-my-namespace-aws-creds created
2023/05/15 18:10:34 Saved credentials configuration to: credrequests-ccoctl-output/manifests/my-namespace-aws-creds-credentials.yaml
2023/05/15 18:10:35 Updated Role policy for Role my-role-my-namespace-aws-creds

```

Copy the **<aws_role_arn>** from the output to use in the next step. For example, **arn:aws:iam::<aws_account_id>:role/my-role-my-namespace-aws-creds**.

- e. Bind the service account with the role ARN by running the following command:

```
$ oc annotate -n my-namespace sa/aws-provider eks.amazonaws.com/role-arn="
<aws_role_arn>"
```

3. Create a secret provider class to define your secrets store provider:

- a. Create a YAML file that defines the **SecretProviderClass** object:

Example secret-provider-class-aws.yaml

```

apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: my-aws-provider
  namespace: my-namespace
spec:
  provider: aws
  parameters:
    objects: |
      - objectName: "testSecret"
        objectType: "secretsmanager"

```

- 1 Specify the name for the secret provider class.
- 2 Specify the namespace for the secret provider class.
- 3 Specify the provider as **aws**.
- 4 Specify the provider-specific configuration parameters.

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

```
$ oc create -f secret-provider-class-aws.yaml
```

4. Create a deployment to use this secret provider class:

- a. Create a YAML file that defines the **Deployment** object:

Example deployment.yaml

```

apiVersion: apps/v1
kind: Deployment
metadata:

```

```

name: my-aws-deployment
namespace: my-namespace
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-storage
  template:
    metadata:
      labels:
        app: my-storage
    spec:
      serviceAccountName: aws-provider
      containers:
        - name: busybox
          image: k8s.gcr.io/e2e-test-images/busybox:1.29
          command:
            - "/bin/sleep"
            - "10000"
          volumeMounts:
            - name: secrets-store-inline
              mountPath: "/mnt/secrets-store"
              readOnly: true
      volumes:
        - name: secrets-store-inline
          csi:
            driver: secrets-store.csi.k8s.io
            readOnly: true
            volumeAttributes:
              secretProviderClass: "my-aws-provider"

```

- 1 Specify the name for the deployment.
- 2 Specify the namespace for the deployment. This must be the same namespace as the secret provider class.
- 3 Specify the name of the secret provider class.

b. Create the **Deployment** object by running the following command:

```
$ oc create -f deployment.yaml
```

Verification

- Verify that you can access the secrets from AWS Secrets Manager in the pod volume mount:
 - a. List the secrets in the pod mount by running the following command:

```
$ oc exec my-aws-deployment-<hash> -n my-namespace -- ls /mnt/secrets-store/
```

Example output

```
testSecret
```

- b. View a secret in the pod mount by running the following command:

```
$ oc exec my-aws-deployment-<hash> -n my-namespace -- cat /mnt/secrets-store/testSecret
```

Example output

```
<secret_value>
```

Additional resources

- [Configuring the Cloud Credential Operator utility](#)

2.8.3.2. Mounting secrets from AWS Systems Manager Parameter Store

You can use the Secrets Store CSI Driver Operator to mount secrets from AWS Systems Manager Parameter Store external secrets store to a Container Storage Interface (CSI) volume in OpenShift Container Platform.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the **jq** tool.
- You have extracted and prepared the **ccocli** utility.
- You have installed the cluster on Amazon Web Services (AWS) and the cluster uses AWS Security Token Service (STS).
- You have installed the Secrets Store CSI Driver Operator. For more information, see "Installing the Secrets Store CSI driver".
- You have configured AWS Systems Manager Parameter Store to store the required secrets.

Procedure

1. Install the AWS Systems Manager Parameter Store provider:
 - a. Create a YAML file by using the following example configuration:



IMPORTANT

The AWS Systems Manager Parameter Store provider for the Secrets Store CSI driver is an upstream provider.

This configuration is modified from the configuration provided in the upstream [AWS documentation](#) so that it works properly with OpenShift Container Platform. Changes to this configuration might impact functionality.

Example aws-provider.yaml file

```
apiVersion: v1
```

```

kind: ServiceAccount
metadata:
  name: csi-secrets-store-provider-aws
  namespace: openshift-cluster-csi-drivers
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: csi-secrets-store-provider-aws-cluster-role
rules:
- apiGroups: [""]
  resources: ["serviceaccounts/token"]
  verbs: ["create"]
- apiGroups: [""]
  resources: ["serviceaccounts"]
  verbs: ["get"]
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get"]
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: csi-secrets-store-provider-aws-cluster-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: csi-secrets-store-provider-aws-cluster-role
subjects:
- kind: ServiceAccount
  name: csi-secrets-store-provider-aws
  namespace: openshift-cluster-csi-drivers
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  namespace: openshift-cluster-csi-drivers
  name: csi-secrets-store-provider-aws
  labels:
    app: csi-secrets-store-provider-aws
spec:
  updateStrategy:
    type: RollingUpdate
  selector:
    matchLabels:
      app: csi-secrets-store-provider-aws
  template:
    metadata:
      labels:
        app: csi-secrets-store-provider-aws
    spec:
      serviceAccountName: csi-secrets-store-provider-aws
      hostNetwork: false

```

```

containers:
  - name: provider-aws-installer
    image: public.ecr.aws/aws-secrets-manager/secrets-store-csi-driver-provider-aws:1.0.r2-50-g5b4aca1-2023.06.09.21.19
    imagePullPolicy: Always
    args:
      - --provider-volume=/etc/kubernetes/secrets-store-csi-providers
    resources:
      requests:
        cpu: 50m
        memory: 100Mi
      limits:
        cpu: 50m
        memory: 100Mi
    securityContext:
      privileged: true
    volumeMounts:
      - mountPath: "/etc/kubernetes/secrets-store-csi-providers"
        name: providervol
      - name: mountpoint-dir
        mountPath: /var/lib/kubelet/pods
        mountPropagation: HostToContainer
    tolerations:
      - operator: Exists
    volumes:
      - name: providervol
        hostPath:
          path: "/etc/kubernetes/secrets-store-csi-providers"
      - name: mountpoint-dir
        hostPath:
          path: /var/lib/kubelet/pods
          type: DirectoryOrCreate
    nodeSelector:
      kubernetes.io/os: linux

```

- b. Grant privileged access to the **csi-secrets-store-provider-aws** service account by running the following command:

```
$ oc adm policy add-scc-to-user privileged -z csi-secrets-store-provider-aws -n openshift-cluster-csi-drivers
```

- c. Create the provider resources by running the following command:

```
$ oc apply -f aws-provider.yaml
```

2. Grant the read permission to the service account for the AWS secret object:

- a. Create a directory to contain the credentials request by running the following command:

```
$ mkdir <aws_creds_directory_name>
```

- b. Create a YAML file that defines the **CredentialsRequest** resource configuration. See the following example configuration:

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

```

kind: CredentialsRequest
metadata:
  name: aws-creds-request
  namespace: openshift-cloud-credential-operator
spec:
  providerSpec:
    apiVersion: cloudcredential.openshift.io/v1
    kind: AWSProviderSpec
    statementEntries:
      - action:
          - "ssm:GetParameter"
          - "ssm:GetParameters"
        effect: Allow
        resource: "arn:*.ssm:*.parameter/testParameter*"
  secretRef:
    name: aws-creds
    namespace: my-namespace
  serviceAccountNames:
    - <service_account_name>

```

- c. Retrieve the OpenID Connect (OIDC) provider by running the following command:

```
$ oc get --raw=/well-known/openid-configuration | jq -r '.issuer'
```

Example output

```
https://<oidc_provider_name>
```

Copy the OIDC provider name **<oidc_provider_name>** from the output to use in the next step.

- d. Use the **ccoctl** tool to process the credentials request by running the following command:

```
$ ccoctl aws create-iam-roles \
  --name my-role --region=<aws_region> \
  --credentials-requests-dir=<aws_creds_dir_name> \
  --identity-provider-arn arn:aws:iam::<aws_account_id>:oidc-
provider/<oidc_provider_name> --output-dir=<output_dir_name>
```

Example output

```

2023/05/15 18:10:34 Role arn:aws:iam::<aws_account_id>:role/my-role-my-namespace-
aws-creds created
2023/05/15 18:10:34 Saved credentials configuration to: credrequests-ccoctl-
output/manifests/my-namespace-aws-creds-credentials.yaml
2023/05/15 18:10:35 Updated Role policy for Role my-role-my-namespace-aws-creds

```

Copy the **<aws_role_arn>** from the output to use in the next step. For example, **arn:aws:iam::<aws_account_id>:role/my-role-my-namespace-aws-creds**.

- e. Bind the service account with the role ARN by running the following command:

```
$ oc annotate -n my-namespace sa/aws-provider eks.amazonaws.com/role-arn="
<aws_role_arn>"
```

3. Create a secret provider class to define your secrets store provider:

- a. Create a YAML file that defines the **SecretProviderClass** object:

Example secret-provider-class-aws.yaml

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: my-aws-provider
  namespace: my-namespace
spec:
  provider: aws
  parameters:
    objects: |
      - objectName: "testParameter"
        objectType: "ssmparameter"
```

- 1 Specify the name for the secret provider class.
- 2 Specify the namespace for the secret provider class.
- 3 Specify the provider as **aws**.
- 4 Specify the provider-specific configuration parameters.

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

```
$ oc create -f secret-provider-class-aws.yaml
```

4. Create a deployment to use this secret provider class:

- a. Create a YAML file that defines the **Deployment** object:

Example deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-aws-deployment
  namespace: my-namespace
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-storage
  template:
    metadata:
      labels:
        app: my-storage
    spec:
      serviceAccountName: aws-provider
```



```

containers:
- name: busybox
  image: k8s.gcr.io/e2e-test-images/busybox:1.29
  command:
  - "/bin/sleep"
  - "10000"
  volumeMounts:
  - name: secrets-store-inline
    mountPath: "/mnt/secrets-store"
    readOnly: true
  volumes:
  - name: secrets-store-inline
    csi:
      driver: secrets-store.csi.k8s.io
      readOnly: true
      volumeAttributes:
        secretProviderClass: "my-aws-provider" 3

```

- 1 Specify the name for the deployment.
- 2 Specify the namespace for the deployment. This must be the same namespace as the secret provider class.
- 3 Specify the name of the secret provider class.

b. Create the **Deployment** object by running the following command:

```
$ oc create -f deployment.yaml
```

Verification

- Verify that you can access the secrets from AWS Systems Manager Parameter Store in the pod volume mount:
 - a. List the secrets in the pod mount by running the following command:

```
$ oc exec my-aws-deployment-<hash> -n my-namespace -- ls /mnt/secrets-store/
```

Example output

```
testParameter
```

b. View a secret in the pod mount by running the following command:

```
$ oc exec my-aws-deployment-<hash> -n my-namespace -- cat /mnt/secrets-store/testSecret
```

Example output

```
<secret_value>
```

Additional resources

- [Configuring the Cloud Credential Operator utility](#)

2.8.3.3. Mounting secrets from Azure Key Vault

You can use the Secrets Store CSI Driver Operator to mount secrets from Microsoft Azure Key Vault to a Container Storage Interface (CSI) volume in OpenShift Container Platform. To mount secrets from Azure Key Vault.

Prerequisites

- You have installed a cluster on Azure.
- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the Azure CLI (**az**).
- You have installed the Secrets Store CSI Driver Operator. See "Installing the Secrets Store CSI driver" for instructions.
- You have configured Azure Key Vault to store the required secrets.

Procedure

1. Install the Azure Key Vault provider:
 - a. Create a YAML file named **azure-provider.yaml** that defines the **ServiceAccount** resource configuration. See the following example configuration:



IMPORTANT

The Azure Key Vault provider for the Secrets Store CSI driver is an upstream provider.

This configuration is modified from the configuration provided in the upstream [Azure documentation](#) so that it works properly with OpenShift Container Platform. Changes to this configuration might impact functionality.

Example azure-provider.yaml file

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: csi-secrets-store-provider-azure
  namespace: openshift-cluster-csi-drivers
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: csi-secrets-store-provider-azure-cluster-role
rules:
- apiGroups: [""]
  resources: ["serviceaccounts/token"]
  verbs: ["create"]
- apiGroups: [""]
  resources: ["serviceaccounts"]
```

```

    verbs: ["get"]
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: csi-secrets-store-provider-azure-cluster-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: csi-secrets-store-provider-azure-cluster-role
subjects:
- kind: ServiceAccount
  name: csi-secrets-store-provider-azure
  namespace: openshift-cluster-csi-drivers
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  namespace: openshift-cluster-csi-drivers
  name: csi-secrets-store-provider-azure
  labels:
    app: csi-secrets-store-provider-azure
spec:
  updateStrategy:
    type: RollingUpdate
  selector:
    matchLabels:
      app: csi-secrets-store-provider-azure
  template:
    metadata:
      labels:
        app: csi-secrets-store-provider-azure
    spec:
      serviceAccountName: csi-secrets-store-provider-azure
      hostNetwork: true
      containers:
        - name: provider-azure-installer
          image: mcr.microsoft.com/oss/azure/secrets-store/provider-azure:v1.4.1
          imagePullPolicy: IfNotPresent
          args:
            - --endpoint=unix:///provider/azure.sock
            - --construct-pem-chain=true
            - --healthz-port=8989
            - --healthz-path=/healthz
            - --healthz-timeout=5s
          livenessProbe:
            httpGet:
              path: /healthz
              port: 8989
            failureThreshold: 3

```

```

    initialDelaySeconds: 5
    timeoutSeconds: 10
    periodSeconds: 30
  resources:
    requests:
      cpu: 50m
      memory: 100Mi
    limits:
      cpu: 50m
      memory: 100Mi
  securityContext:
    allowPrivilegeEscalation: false
    readOnlyRootFilesystem: true
    runAsUser: 0
  capabilities:
    drop:
      - ALL
  volumeMounts:
    - mountPath: "/provider"
      name: providervol
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: type
                operator: NotIn
                values:
                  - virtual-kubelet
  volumes:
    - name: providervol
      hostPath:
        path: "/var/run/secrets-store-csi-providers"
  tolerations:
    - operator: Exists
  nodeSelector:
    kubernetes.io/os: linux

```

- b. Grant privileged access to the **csi-secrets-store-provider-azure** service account by running the following command:

```
$ oc adm policy add-scc-to-user privileged -z csi-secrets-store-provider-azure -n openshift-cluster-csi-drivers
```

- c. Create the provider resources by running the following command:

```
$ oc apply -f azure-provider.yaml
```

2. Create a service principal to access the key vault:

- a. Set the service principal client secret as an environment variable by running the following command:

```
$ SERVICE_PRINCIPAL_CLIENT_SECRET="$(az ad sp create-for-rbac --name https://$KEYVAULT_NAME --query 'password' -otsv)"
```

- b. Set the service principal client ID as an environment variable by running the following command:

```
$ SERVICE_PRINCIPAL_CLIENT_ID=$(az ad sp list --display-name
https://$KEYVAULT_NAME --query '[0].appId' -otsv)"
```

- c. Create a generic secret with the service principal client secret and ID by running the following command:

```
$ oc create secret generic secrets-store-creds -n my-namespace --from-literal
clientid=${SERVICE_PRINCIPAL_CLIENT_ID} --from-literal
clientsecret=${SERVICE_PRINCIPAL_CLIENT_SECRET}
```

- d. Apply the **secrets-store.csi.k8s.io/used=true** label to allow the provider to find this **nodePublishSecretRef** secret:

```
$ oc -n my-namespace label secret secrets-store-creds secrets-
store.csi.k8s.io/used=true
```

3. Create a secret provider class to define your secrets store provider:

- a. Create a YAML file that defines the **SecretProviderClass** object:

Example secret-provider-class-azure.yaml

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: my-azure-provider
  namespace: my-namespace
spec:
  provider: azure
  parameters:
    usePodIdentity: "false"
    useVMManagedIdentity: "false"
    userAssignedIdentityID: ""
    keyvaultName: "kvname"
    objects: |
      array:
        - |
          objectName: secret1
          objectType: secret
  tenantId: "tid"
```

- 1 Specify the name for the secret provider class.
- 2 Specify the namespace for the secret provider class.
- 3 Specify the provider as **azure**.
- 4 Specify the provider-specific configuration parameters.

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

```
$ oc create -f secret-provider-class-azure.yaml
```

4. Create a deployment to use this secret provider class:

a. Create a YAML file that defines the **Deployment** object:

Example deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-azure-deployment 1
  namespace: my-namespace 2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-storage
  template:
    metadata:
      labels:
        app: my-storage
    spec:
      containers:
        - name: busybox
          image: k8s.gcr.io/e2e-test-images/busybox:1.29
          command:
            - "/bin/sleep"
            - "10000"
          volumeMounts:
            - name: secrets-store-inline
              mountPath: "/mnt/secrets-store"
              readOnly: true
      volumes:
        - name: secrets-store-inline
          csi:
            driver: secrets-store.csi.k8s.io
            readOnly: true
            volumeAttributes:
              secretProviderClass: "my-azure-provider" 3
            nodePublishSecretRef:
              name: secrets-store-creds 4
```

- 1 Specify the name for the deployment.
- 2 Specify the namespace for the deployment. This must be the same namespace as the secret provider class.
- 3 Specify the name of the secret provider class.
- 4 Specify the name of the Kubernetes secret that contains the service principal credentials to access Azure Key Vault.

b. Create the **Deployment** object by running the following command:

```
$ oc create -f deployment.yaml
```

Verification

- Verify that you can access the secrets from Azure Key Vault in the pod volume mount:
 - a. List the secrets in the pod mount by running the following command:

```
$ oc exec my-azure-deployment-<hash> -n my-namespace -- ls /mnt/secrets-store/
```

Example output

```
secret1
```

- b. View a secret in the pod mount by running the following command:

```
$ oc exec my-azure-deployment-<hash> -n my-namespace -- cat /mnt/secrets-store/secret1
```

Example output

```
my-secret-value
```

2.8.3.4. Mounting secrets from Google Secret Manager

You can use the Secrets Store CSI Driver Operator to mount secrets from Google Secret Manager to a Container Storage Interface (CSI) volume in OpenShift Container Platform. To mount secrets from Google Secret Manager, your cluster must be installed on Google Cloud.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the Secrets Store CSI Driver Operator. See "Installing the Secrets Store CSI driver" for instructions.
- You have configured Google Secret Manager to store the required secrets.
- You have created a service account key named **key.json** from your Google Cloud service account.

Procedure

1. Install the Google Secret Manager provider:
 - a. Create a YAML file Create a YAML file named **gcp-provider.yaml** that defines the **ServiceAccount** resource configuration. See the following example configuration:

Example gcp-provider.yaml file

```
apiVersion: v1
kind: ServiceAccount
```

```

metadata:
  name: csi-secrets-store-provider-gcp
  namespace: openshift-cluster-csi-drivers
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: csi-secrets-store-provider-gcp-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: csi-secrets-store-provider-gcp-role
subjects:
- kind: ServiceAccount
  name: csi-secrets-store-provider-gcp
  namespace: openshift-cluster-csi-drivers
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: csi-secrets-store-provider-gcp-role
rules:
- apiGroups:
  - ""
  resources:
  - serviceaccounts/token
  verbs:
  - create
- apiGroups:
  - ""
  resources:
  - serviceaccounts
  verbs:
  - get
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: csi-secrets-store-provider-gcp
  namespace: openshift-cluster-csi-drivers
  labels:
    app: csi-secrets-store-provider-gcp
spec:
  updateStrategy:
    type: RollingUpdate
  selector:
    matchLabels:
      app: csi-secrets-store-provider-gcp
  template:
    metadata:
      labels:
        app: csi-secrets-store-provider-gcp
    spec:
      serviceAccountName: csi-secrets-store-provider-gcp
      initContainers:
      - name: chown-provider-mount

```



```

image: busybox
command:
- chown
- "1000:1000"
- /etc/kubernetes/secrets-store-csi-providers
volumeMounts:
- mountPath: "/etc/kubernetes/secrets-store-csi-providers"
  name: providervol
securityContext:
  privileged: true
hostNetwork: false
hostPID: false
hostIPC: false
containers:
- name: provider
  image: us-docker.pkg.dev/secretmanager-csi/secrets-store-csi-driver-provider-
gcp/plugin@sha256:a493a78bbb4ebce5f5de15acdccc6f4d19486eae9aa4fa529bb60ac112
dd6650
  securityContext:
    privileged: true
  imagePullPolicy: IfNotPresent
  resources:
    requests:
      cpu: 50m
      memory: 100Mi
    limits:
      cpu: 50m
      memory: 100Mi
  env:
    - name: TARGET_DIR
      value: "/etc/kubernetes/secrets-store-csi-providers"
  volumeMounts:
    - mountPath: "/etc/kubernetes/secrets-store-csi-providers"
      name: providervol
      mountPropagation: None
      readOnly: false
  livenessProbe:
    failureThreshold: 3
    httpGet:
      path: /live
      port: 8095
    initialDelaySeconds: 5
    timeoutSeconds: 10
    periodSeconds: 30
  volumes:
    - name: providervol
      hostPath:
        path: /etc/kubernetes/secrets-store-csi-providers
  tolerations:
    - key: kubernetes.io/arch
      operator: Equal
      value: amd64
      effect: NoSchedule
  nodeSelector:
    kubernetes.io/os: linux

```

- b. Grant privileged access to the **csi-secrets-store-provider-gcp** service account by running the following command:

```
$ oc adm policy add-scc-to-user privileged -z csi-secrets-store-provider-gcp -n openshift-cluster-csi-drivers
```

- c. Create the provider resources by running the following command:

```
$ oc apply -f gcp-provider.yaml
```

2. Grant a read permission to the Google Secret Manager secret:

- a. Create a new project by running the following command:

```
$ oc new-project my-namespace
```

- b. Label the **my-namespace** namespace for pod security admission by running the following command:

```
$ oc label ns my-namespace security.openshift.io/scc.podSecurityLabelSync=false pod-security.kubernetes.io/enforce=privileged pod-security.kubernetes.io/audit=privileged pod-security.kubernetes.io/warn=privileged --overwrite
```

- c. Create a service account for the pod deployment:

```
$ oc create serviceaccount my-service-account --namespace=my-namespace
```

- d. Create a generic secret from the **key.json** file by running the following command:

```
$ oc create secret generic secrets-store-creds -n my-namespace --from-file=key.json 1
```

1 You created this **key.json** file from the Google Secret Manager.

- e. Apply the **secrets-store.csi.k8s.io/used=true** label to allow the provider to find this **nodePublishSecretRef** secret:

```
$ oc -n my-namespace label secret secrets-store-creds secrets-store.csi.k8s.io/used=true
```

3. Create a secret provider class to define your secrets store provider:

- a. Create a YAML file that defines the **SecretProviderClass** object:

Example **secret-provider-class-gcp.yaml**

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: my-gcp-provider 1
  namespace: my-namespace 2
spec:
  provider: gcp 3
```

```
parameters:
  secrets: |
    - resourceName: "projects/my-project/secrets/testsecret1/versions/1"
      path: "testsecret1.txt"
```

- 1 Specify the name for the secret provider class.
- 2 Specify the namespace for the secret provider class.
- 3 Specify the provider as **gcp**.
- 4 Specify the provider-specific configuration parameters.

b. Create the **SecretProviderClass** object by running the following command:

```
$ oc create -f secret-provider-class-gcp.yaml
```

4. Create a deployment to use this secret provider class:

a. Create a YAML file that defines the **Deployment** object:

Example deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-gcp-deployment
  namespace: my-namespace
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-storage
  template:
    metadata:
      labels:
        app: my-storage
    spec:
      serviceAccountName: my-service-account
      containers:
        - name: busybox
          image: k8s.gcr.io/e2e-test-images/busybox:1.29
          command:
            - "/bin/sleep"
            - "10000"
          volumeMounts:
            - name: secrets-store-inline
              mountPath: "/mnt/secrets-store"
              readOnly: true
      volumes:
        - name: secrets-store-inline
          csi:
            driver: secrets-store.csi.k8s.io
            readOnly: true
```

```

volumeAttributes:
  secretProviderClass: "my-gcp-provider" 4
  nodePublishSecretRef:
    name: secrets-store-creds 5

```

- 1 Specify the name for the deployment.
- 2 Specify the namespace for the deployment. This must be the same namespace as the secret provider class.
- 3 Specify the service account you created.
- 4 Specify the name of the secret provider class.
- 5 Specify the name of the Kubernetes secret that contains the service principal credentials to access Google Secret Manager.

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

```
$ oc create -f deployment.yaml
```

Verification

- Verify that you can access the secrets from Google Secret Manager in the pod volume mount:
 - a. List the secrets in the pod mount by running the following command:

```
$ oc exec my-gcp-deployment-<hash> -n my-namespace -- ls /mnt/secrets-store/
```

Example output

```
testsecret1
```

- b. View a secret in the pod mount by running the following command:

```
$ oc exec my-gcp-deployment-<hash> -n my-namespace -- cat /mnt/secrets-store/testsecret1
```

Example output

```
<secret_value>
```

2.8.3.5. Mounting secrets from HashiCorp Vault

You can use the Secrets Store CSI Driver Operator to mount secrets from HashiCorp Vault to a Container Storage Interface (CSI) volume in OpenShift Container Platform.



IMPORTANT

Mounting secrets from HashiCorp Vault by using the Secrets Store CSI Driver Operator has been tested with the following cloud providers:

- Amazon Web Services (AWS)
- Microsoft Azure

Other cloud providers might work, but have not been tested yet. Additional cloud providers might be tested in the future.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the Secrets Store CSI Driver Operator. See "Installing the Secrets Store CSI driver" for instructions.
- You have installed Helm.

Procedure

1. Add the HashiCorp Helm repository by running the following command:

```
$ helm repo add hashicorp https://helm.releases.hashicorp.com
```

2. Update all repositories to ensure that Helm is aware of the latest versions by running the following command:

```
$ helm repo update
```

3. Install the HashiCorp Vault provider:

- a. Create a new project for Vault by running the following command:

```
$ oc new-project vault
```

- b. Label the **vault** namespace for pod security admission by running the following command:

```
$ oc label ns vault security.openshift.io/scc.podSecurityLabelSync=false pod-security.kubernetes.io/enforce=privileged pod-security.kubernetes.io/audit=privileged pod-security.kubernetes.io/warn=privileged --overwrite
```

- c. Grant privileged access to the **vault** service account by running the following command:

```
$ oc adm policy add-scc-to-user privileged -z vault -n vault
```

- d. Grant privileged access to the **vault-csi-provider** service account by running the following command:

```
$ oc adm policy add-scc-to-user privileged -z vault-csi-provider -n vault
```

- e. Deploy HashiCorp Vault by running the following command:

```
$ helm install vault hashicorp/vault --namespace=vault \
--set "server.dev.enabled=true" \
--set "injector.enabled=false" \
--set "csi.enabled=true" \
--set "global.openshift=true" \
--set "injector.agentImage.repository=docker.io/hashicorp/vault" \
--set "server.image.repository=docker.io/hashicorp/vault" \
--set "csi.image.repository=docker.io/hashicorp/vault-csi-provider" \
--set "csi.agent.image.repository=docker.io/hashicorp/vault" \
--set "csi.daemonSet.providersDir=/var/run/secrets-store-csi-providers"
```

- f. Patch the **vault-csi-driver** daemon set to set the **securityContext** to **privileged** by running the following command:

```
$ oc patch daemonset -n vault vault-csi-provider --type='json' -p='[{"op": "add", "path":
"/spec/template/spec/containers/0/securityContext", "value": {"privileged": true} }]'
```

- g. Verify that the **vault-csi-provider** pods have started properly by running the following command:

```
$ oc get pods -n vault
```

Example output

NAME	READY	STATUS	RESTARTS	AGE
vault-0	1/1	Running	0	24m
vault-csi-provider-87rgw	1/2	Running	0	5s
vault-csi-provider-bd6hp	1/2	Running	0	4s
vault-csi-provider-smlv7	1/2	Running	0	5s

4. Configure HashiCorp Vault to store the required secrets:

- a. Create a secret by running the following command:

```
$ oc exec vault-0 --namespace=vault -- vault kv put secret/example1 testSecret1=my-
secret-value
```

- b. Verify that the secret is readable at the path **secret/example1** by running the following command:

```
$ oc exec vault-0 --namespace=vault -- vault kv get secret/example1
```

Example output

```
= Secret Path =
secret/data/example1

===== Metadata =====
Key          Value
---          -
created_time  2024-04-05T07:05:16.713911211Z
custom_metadata  <nil>
deletion_time  n/a
```

```

destroyed      false
version        1

=== Data ===
Key            Value
---          ---
testSecret1    my-secret-value

```

5. Configure Vault to use Kubernetes authentication:

- a. Enable the Kubernetes auth method by running the following command:

```
$ oc exec vault-0 --namespace=vault -- vault auth enable kubernetes
```

Example output

```
Success! Enabled kubernetes auth method at: kubernetes/
```

- b. Configure the Kubernetes auth method:

- i. Set the token reviewer as an environment variable by running the following command:

```
$ TOKEN_REVIEWER_JWT="$(oc exec vault-0 --namespace=vault -- cat
/var/run/secrets/kubernetes.io/serviceaccount/token)"
```

- ii. Set the Kubernetes service IP address as an environment variable by running the following command:

```
$ KUBERNETES_SERVICE_IP="$(oc get svc kubernetes --namespace=default -o
go-template="{{ .spec.clusterIP }}")"
```

- iii. Update the Kubernetes auth method by running the following command:

```
$ oc exec -i vault-0 --namespace=vault -- vault write auth/kubernetes/config \
issuer="https://kubernetes.default.svc.cluster.local" \
token_reviewer_jwt="{TOKEN_REVIEWER_JWT}" \
kubernetes_host="https://{KUBERNETES_SERVICE_IP}:443" \
kubernetes_ca_cert=@/var/run/secrets/kubernetes.io/serviceaccount/ca.crt
```

Example output

```
Success! Data written to: auth/kubernetes/config
```

- c. Create a policy for the application by running the following command:

```
$ oc exec -i vault-0 --namespace=vault -- vault policy write csi -<<EOF
path "secret/data/*" {
  capabilities = ["read"]
}
EOF
```

Example output

```
Success! Uploaded policy: csi
```

- d. Create an authentication role to access the application by running the following command:

```
$ oc exec -i vault-0 --namespace=vault -- vault write auth/kubernetes/role/csi \
  bound_service_account_names=default \
  bound_service_account_namespaces=default,test-ns,negative-test-ns,my-namespace \
  policies=csi \
  ttl=20m
```

Example output

```
Success! Data written to: auth/kubernetes/role/csi
```

6. Create a secret provider class to define your secrets store provider:
- a. Create a YAML file that defines the **SecretProviderClass** object:

Example secret-provider-class-vault.yaml

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: my-vault-provider
  namespace: my-namespace
spec:
  provider: vault
  parameters:
    roleName: "csi"
    vaultAddress: "http://vault.vault:8200"
    objects: |
      - secretPath: "secret/data/example1"
        objectName: "testSecret1"
        secretKey: "testSecret1"
```

- 1 Specify the name for the secret provider class.
- 2 Specify the namespace for the secret provider class.
- 3 Specify the provider as **vault**.
- 4 Specify the provider-specific configuration parameters.

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

```
$ oc create -f secret-provider-class-vault.yaml
```

7. Create a deployment to use this secret provider class:
- a. Create a YAML file that defines the **Deployment** object:

Example deployment.yaml


```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox-deployment
  namespace: my-namespace
  labels:
    app: busybox
spec:
  replicas: 1
  selector:
    matchLabels:
      app: busybox
  template:
    metadata:
      labels:
        app: busybox
    spec:
      terminationGracePeriodSeconds: 0
      containers:
        - image: registry.k8s.io/e2e-test-images/busybox:1.29-4
          name: busybox
          imagePullPolicy: IfNotPresent
          command:
            - "/bin/sleep"
            - "10000"
          volumeMounts:
            - name: secrets-store-inline
              mountPath: "/mnt/secrets-store"
              readOnly: true
      volumes:
        - name: secrets-store-inline
          csi:
            driver: secrets-store.csi.k8s.io
            readOnly: true
            volumeAttributes:
              secretProviderClass: "my-vault-provider"

```

- 1 Specify the name for the deployment.
- 2 Specify the namespace for the deployment. This must be the same namespace as the secret provider class.
- 3 Specify the name of the secret provider class.

b. Create the **Deployment** object by running the following command:

```
$ oc create -f deployment.yaml
```

Verification

- a. Verify that all of the **vault** pods are running properly by running the following command:

```
$ oc get pods -n vault
```

Example output

NAME	READY	STATUS	RESTARTS	AGE
vault-0	1/1	Running	0	43m
vault-csi-provider-87rgw	2/2	Running	0	19m
vault-csi-provider-bd6hp	2/2	Running	0	19m
vault-csi-provider-smlv7	2/2	Running	0	19m

- b. Verify that all of the **secrets-store-csi-driver** pods are running by running the following command:

```
$ oc get pods -n openshift-cluster-csi-drivers | grep -E "secrets"
```

Example output

secrets-store-csi-driver-node-46d2g	3/3	Running	0	45m
secrets-store-csi-driver-node-d2jjn	3/3	Running	0	45m
secrets-store-csi-driver-node-drmt4	3/3	Running	0	45m
secrets-store-csi-driver-node-j2wlt	3/3	Running	0	45m
secrets-store-csi-driver-node-v9xv4	3/3	Running	0	45m
secrets-store-csi-driver-node-vlz28	3/3	Running	0	45m
secrets-store-csi-driver-operator-84bd699478-fpxrw	1/1	Running	0	47m

1. Verify that you can access the secrets from your HashiCorp Vault in the pod volume mount:
- c. List the secrets in the pod mount by running the following command:

```
$ oc exec busybox-deployment-<hash> -n my-namespace -- ls /mnt/secrets-store/
```

Example output

```
testSecret1
```

- d. View a secret in the pod mount by running the following command:

```
$ oc exec busybox-deployment-<hash> -n my-namespace -- cat /mnt/secrets-store/testSecret1
```

Example output

```
my-secret-value
```

Additional resources

- [Installing Helm](#)

2.8.4. Enabling synchronization of mounted content as Kubernetes secrets

You can enable synchronization to create Kubernetes secrets from the content on a mounted volume. An example where you might want to enable synchronization is to use an environment variable in your deployment to reference the Kubernetes secret.



WARNING

Do not enable synchronization if you do not want to store your secrets on your OpenShift Container Platform cluster and in etcd. Enable this functionality only if you require it, such as when you want to use environment variables to refer to the secret.

If you enable synchronization, the secrets from the mounted volume are synchronized as Kubernetes secrets after you start a pod that mounts the secrets.

The synchronized Kubernetes secret is deleted when all pods that mounted the content are deleted.

Prerequisites

- You have installed the Secrets Store CSI Driver Operator.
- You have installed a secrets store provider.
- You have created the secret provider class.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Edit the **SecretProviderClass** resource by running the following command:

```
$ oc edit secretproviderclass my-azure-provider 1
```

- 1 Replace **my-azure-provider** with the name of your secret provider class.

2. Add the **secretsObjects** section with the configuration for the synchronized Kubernetes secrets:

```
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: my-azure-provider
  namespace: my-namespace
spec:
  provider: azure
  secretObjects: 1
    - secretName: tlssecret 2
      type: kubernetes.io/tls 3
      labels:
        environment: "test"
      data:
        - objectName: tlskey 4
          key: tls.key 5
        - objectName: tlscrt
```

```

    key: tls.crt
  parameters:
    usePodIdentity: "false"
    keyvaultName: "kvname"
  objects: |
    array:
      - |
        objectName: tlskey
        objectType: secret
      - |
        objectName: tlscrt
        objectType: secret
  tenantId: "tid"

```

- 1 Specify the configuration for synchronized Kubernetes secrets.
- 2 Specify the name of the Kubernetes **Secret** object to create.
- 3 Specify the type of Kubernetes **Secret** object to create. For example, **Opaque** or **kubernetes.io/tls**.
- 4 Specify the object name or alias of the mounted content to synchronize.
- 5 Specify the data field from the specified **objectName** to populate the Kubernetes secret with.

3. Save the file to apply the changes.

2.8.5. Viewing the status of secrets in the pod volume mount

You can view detailed information, including the versions, of the secrets in the pod volume mount.

The Secrets Store CSI Driver Operator creates a **SecretProviderClassPodStatus** resource in the same namespace as the pod. You can review this resource to see detailed information, including versions, about the secrets in the pod volume mount.

Prerequisites

- You have installed the Secrets Store CSI Driver Operator.
- You have installed a secrets store provider.
- You have created the secret provider class.
- You have deployed a pod that mounts a volume from the Secrets Store CSI Driver Operator.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

- View detailed information about the secrets in a pod volume mount by running the following command:

```
$ oc get secretproviderclasspodstatus <secret_provider_class_pod_status_name> -o yaml
```

1

- 1 The name of the secret provider class pod status object is in the format of **<pod_name>-<namespace>-<secret_provider_class_name>**.

Example output

```
...
status:
  mounted: true
  objects:
    - id: secret/tlsrct
      version: f352293b97da4fa18d96a9528534cb33
    - id: secret/tlskey
      version: 02534bc3d5df481cb138f8b2a13951ef
  podName: busybox-<hash>
  secretProviderClassName: my-azure-provider
  targetPath: /var/lib/kubelet/pods/f0d49c1e-c87a-4beb-888f-
37798456a3e7/volumes/kubernetes.io~csi/secrets-store-inline/mount
```

2.8.6. Uninstalling the Secrets Store CSI Driver Operator

Prerequisites

- Access to the OpenShift Container Platform web console.
- Administrator access to the cluster.

Procedure

To uninstall the Secrets Store CSI Driver Operator:

1. Stop all application pods that use the **secrets-store.csi.k8s.io** provider.
2. Remove any third-party provider plug-in for your chosen secret store.
3. Remove the Container Storage Interface (CSI) driver and associated manifests:
 - a. Click **Administration** → **CustomResourceDefinitions** → **ClusterCSIDriver**.
 - b. On the **Instances** tab, for **secrets-store.csi.k8s.io**, on the far left side, click the drop-down menu, and then click **Delete ClusterCSIDriver**.
 - c. When prompted, click **Delete**.
4. Verify that the CSI driver pods are no longer running.
5. Uninstall the Secrets Store CSI Driver Operator:



NOTE

Before you can uninstall the Operator, you must remove the CSI driver first.

- a. Click **Ecosystem** → **Installed Operators**.
- b. On the **Installed Operators** page, scroll or type "Secrets Store CSI" into the **Search by name** box to find the Operator, and then click it.
- c. On the upper, right of the **Installed Operators > Operator details** page, click **Actions** → **Uninstall Operator**.
- d. When prompted on the **Uninstall Operator** window, click the **Uninstall** button to remove the Operator from the namespace. Any applications deployed by the Operator on the cluster need to be cleaned up manually.
After uninstalling, the Secrets Store CSI Driver Operator is no longer listed in the **Installed Operators** section of the web console.

2.9. AUTHENTICATING PODS WITH SHORT-TERM CREDENTIALS

Some OpenShift Container Platform clusters use [short-term security credentials for individual components](#) that are created and managed outside the cluster. Applications in customer workloads on these clusters can authenticate by using the short-term authentication method that the cluster uses.

2.9.1. Configuring short-term authentication for workloads

To use this authentication method in your applications, you must complete the following steps:

1. Create a federated identity service account in the Identity and Access Management (IAM) settings for your cloud provider.
2. Create an OpenShift Container Platform service account that can impersonate a service account for your cloud provider.
3. Configure any workloads related to your application to use the OpenShift Container Platform service account.

2.9.1.1. Environment and user access requirements

To configure this authentication method, you must meet the following requirements:

- Your cluster must use [short-term security credentials](#).
- You must have access to the OpenShift CLI (**oc**) as a user with the **cluster-admin** role.
- In your cloud provider console, you must have access as a user with privileges to manage Identity and Access Management (IAM) and federated identity configurations.

2.9.2. Configuring GCP Workload Identity authentication for applications on Google Cloud

To use short-term authentication for applications on a Google Cloud clusters that use GCP Workload Identity authentication, you must complete the following steps:

1. [Configure access in Google Cloud](#).
2. [Create an OpenShift Container Platform service account that can use this access](#).
3. [Deploy customer workloads that authenticate with GCP Workload Identity](#).

2.9.2.1. Creating a federated Google Cloud service account

You can use the Google Cloud console to create a workload identity pool and provider and allow an OpenShift Container Platform service account to impersonate a Google Cloud service account.

Prerequisites

- Your Google Cloud cluster uses GCP Workload Identity.
- You have access to the Google Cloud console as a user with privileges to manage Identity and Access Management (IAM) and workload identity configurations.
- You have created a Google Cloud project to use with your application.

Procedure

1. In the IAM configuration for your Google Cloud project, identify the identity pool and provider that the cluster uses for GCP Workload Identity authentication.
2. Grant permission for external identities to impersonate a Google Cloud service account. With these permissions, an OpenShift Container Platform service account can work as a federated workload identity.
For more information, see Google Cloud documentation about [allowing your external workload to access Google Cloud resources](#).

2.9.2.2. Creating an OpenShift Container Platform service account for Google Cloud

You create an OpenShift Container Platform service account and annotate it to impersonate a Google Cloud service account.

Prerequisites

- Your Google Cloud cluster uses GCP Workload Identity.
- You have created a federated Google Cloud service account.
- You have access to the OpenShift CLI (**oc**) as a user with the **cluster-admin** role.
- You have access to the Google Cloud CLI (**gcloud**) as a user with privileges to manage Identity and Access Management (IAM) and workload identity configurations.

Procedure

1. Create an OpenShift Container Platform service account to use for GCP Workload Identity pod authentication by running the following command:

```
$ oc create serviceaccount <service_account_name>
```

2. Annotate the service account with the identity provider and Google Cloud service account to impersonate by running the following command:

```
$ oc patch serviceaccount <service_account_name> -p '{"metadata": {"annotations": {"cloud.google.com/workload-identity-provider": "projects/<project_number>/locations/global/workloadIdentityPools/<identity_pool>/providers/<identity_provider>"}}}'
```

-

Replace **<project_number>**, **<identity_pool>**, and **<identity_provider>** with the values for your configuration.



NOTE

For **<project_number>**, specify the Google Cloud project number, not the project ID.

3. Annotate the service account with the email address for the Google Cloud service account by running the following command:

```
$ oc patch serviceaccount <service_account_name> -p '{"metadata": {"annotations": {"cloud.google.com/service-account-email": "<service_account_email>"}}}'
```

Replace **<service_account_email>** with the email address for the Google Cloud service account.

TIP

Google Cloud service account email addresses typically use the format **<service_account_name>@<project_id>.iam.gserviceaccount.com**

4. Annotate the service account to use the **direct** external credentials configuration injection mode by running the following command:

```
$ oc patch serviceaccount <service_account_name> -p '{"metadata": {"annotations": {"cloud.google.com/injection-mode": "direct"}}}'
```

In this mode, the Workload Identity Federation webhook controller directly generates the Google Cloud external credentials configuration and injects them into the pod.

5. Use the Google Cloud CLI (**gcloud**) to specify the permissions for the workload by running the following command:

```
$ gcloud projects add-iam-policy-binding <project_id> --member "<service_account_email>" -role "projects/<project_id>/roles/<role_for_workload_permissions>"
```

Replace **<role_for_workload_permissions>** with the role for the workload. Specify a role that grants the permissions that your workload requires.

Verification

- To verify the service account configuration, inspect the **ServiceAccount** manifest by running the following command:

```
$ oc get serviceaccount <service_account_name>
```

In the following example, the **service-a/app-x** OpenShift Container Platform service account can impersonate a Google Cloud service account called **app-x**:

Example output

■


```

apiVersion: v1
kind: ServiceAccount
metadata:
  name: app-x
  namespace: service-a
  annotations:
    cloud.google.com/workload-identity-provider:
"projects/<project_number>/locations/global/workloadIdentityPools/<identity_pool>/providers/<ic
entity_provider>" ❶
    cloud.google.com/service-account-email: "app-x@project.iam.googleapis.com"
    cloud.google.com/audience: "sts.googleapis.com" ❷
    cloud.google.com/token-expiration: "86400" ❸
    cloud.google.com/gcloud-run-as-user: "1000"
    cloud.google.com/injection-mode: "direct" ❹

```

- ❶ The workload identity provider for the service account of the cluster.
- ❷ The allowed audience for the workload identity provider.
- ❸ The token expiration time period in seconds.
- ❹ The **direct** external credentials configuration injection mode.

2.9.2.3. Deploying customer workloads that authenticate with GCP Workload Identity

To use short-term authentication in your application, you must configure its related pods to use the OpenShift Container Platform service account. Use of the OpenShift Container Platform service account triggers the webhook to mutate the pods so they can impersonate the Google Cloud service account.

The following example demonstrates how to deploy a pod that uses the OpenShift Container Platform service account and verify the configuration.

Prerequisites

- Your Google Cloud cluster uses GCP Workload Identity.
- You have created a federated Google Cloud service account.
- You have created an OpenShift Container Platform service account for Google Cloud.

Procedure

1. To create a pod that authenticates with GCP Workload Identity, create a deployment YAML file similar to the following example:

Sample deployment

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: ubi9
spec:
  replicas: 1

```

```

selector:
  matchLabels:
    app: ubi9
template:
  metadata:
    labels:
      app: ubi9
  spec:
    serviceAccountName: "<service_account_name>" ❶
    containers:
      - name: ubi
        image: 'registry.access.redhat.com/ubi9/ubi-micro:latest'
        command:
          - /bin/sh
          - '-c'
          - |
            sleep infinity

```

- ❶ Specify the name of the OpenShift Container Platform service account.

2. Apply the deployment file by running the following command:

```
$ oc apply -f deployment.yaml
```

Verification

- To verify that a pod is using short-term authentication, run the following command:

```
$ oc get pods -o json | jq -r '.items[0].spec.containers[0].env[] |
select(.name=="GOOGLE_APPLICATION_CREDENTIALS")'
```

Example output

```
{ "name": "GOOGLE_APPLICATION_CREDENTIALS", "value": "/var/run/secrets/workload-
identity/federation.json" }
```

The presence of the **GOOGLE_APPLICATION_CREDENTIALS** environment variable indicates a pod that authenticates with GCP Workload Identity.

- To verify additional configuration details, examine the pod specification. The following example pod specifications show the environment variables and volume fields that the webhook mutates.

Example pod specification with the direct injection mode:

```

apiVersion: v1
kind: Pod
metadata:
  name: app-x-pod
  namespace: service-a
annotations:
  cloud.google.com/skip-containers: "init-first,sidecar"
  cloud.google.com/external-credentials-json: |- ❶

```

```

{
  "type": "external_account",
  "audience":
    "//iam.googleapis.com/projects/<project_number>/locations/global/workloadIdentityPools/on-
    prem-kubernetes/providers/<identity_provider>",
  "subject_token_type": "urn:ietf:params:oauth:token-type:jwt",
  "token_url": "https://sts.googleapis.com/v1/token",
  "service_account_impersonation_url":
    "https://iamcredentials.googleapis.com/v1/projects/-/serviceAccounts/app-
    x@project.iam.gserviceaccount.com:generateAccessToken",
  "credential_source": {
    "file": "/var/run/secrets/sts.googleapis.com/serviceaccount/token",
    "format": {
      "type": "text"
    }
  }
}
}
spec:
  serviceAccountName: app-x
  initContainers:
  - name: init-first
    image: container-image:version
  containers:
  - name: sidecar
    image: container-image:version
  - name: container-name
    image: container-image:version
  env: ❷
  - name: GOOGLE_APPLICATION_CREDENTIALS
    value: /var/run/secrets/gcloud/config/federation.json
  - name: CLOUDSDK_COMPUTE_REGION
    value: asia-northeast1
  volumeMounts:
  - name: gcp-iam-token
    readOnly: true
    mountPath: /var/run/secrets/sts.googleapis.com/serviceaccount
  - mountPath: /var/run/secrets/gcloud/config
    name: external-credential-config
    readOnly: true
  volumes:
  - name: gcp-iam-token
    projected:
      sources:
      - serviceAccountToken:
          audience: sts.googleapis.com
          expirationSeconds: 86400
          path: token
  - downwardAPI:
      defaultMode: 288
      items:
      - fieldRef:
          apiVersion: v1
          fieldPath: metadata.annotations['cloud.google.com/external-credentials-json']
        path: federation.json
        name: external-credential-config

```

- 1 The external credentials configuration generated by the webhook controller. The Kubernetes **downwardAPI** volume mounts the configuration into the container filesystem.
- 2 The webhook-injected environment variables for token-based authentication.

2.10. CREATING AND USING CONFIG MAPS

The following sections define config maps and how to create and use them.

2.10.1. Understanding config maps

Many applications require configuration by using some combination of configuration files, command-line arguments, and environment variables. In OpenShift Container Platform, these configuration artifacts are decoupled from image content to keep containerized applications portable.

The **ConfigMap** object provides mechanisms to inject containers with configuration data while keeping containers agnostic of OpenShift Container Platform. A config map can be used to store fine-grained information like individual properties or coarse-grained information like entire configuration files or JSON blobs.

The **ConfigMap** object holds key-value pairs of configuration data that can be consumed in pods or used to store configuration data for system components such as controllers. For example:

ConfigMap Object Definition

```
kind: ConfigMap
apiVersion: v1
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: example-config
  namespace: my-namespace
data: 1
  example.property.1: hello
  example.property.2: world
  example.property.file: |-
    property.1=value-1
    property.2=value-2
    property.3=value-3
binaryData:
  bar: L3Jvb3QvMTAw 2
```

- 1 Contains the configuration data.
- 2 Points to a file that contains non-UTF8 data, for example, a binary Java keystore file. Enter the file data in Base 64.



NOTE

You can use the **binaryData** field when you create a config map from a binary file, such as an image.

Configuration data can be consumed in pods in a variety of ways. A config map can be used to:

- Populate environment variable values in containers
- Set command-line arguments in a container
- Populate configuration files in a volume

Users and system components can store configuration data in a config map.

A config map is similar to a secret, but designed to more conveniently support working with strings that do not contain sensitive information.

2.10.1.1. Config map restrictions

A config map must be created before its contents can be consumed in pods.

Controllers can be written to tolerate missing configuration data. Consult individual components configured by using config maps on a case-by-case basis.

ConfigMap objects reside in a project.

They can only be referenced by pods in the same project.

The Kubelet only supports the use of a config map for pods it gets from the API server.

This includes any pods created by using the CLI, or indirectly from a replication controller. It does not include pods created by using the OpenShift Container Platform node's **--manifest-url** flag, its **--config** flag, or its REST API because these are not common ways to create pods.

2.10.2. Creating a config map in the OpenShift Container Platform web console

You can create a config map in the OpenShift Container Platform web console.

Procedure

- To create a config map as a cluster administrator:
 1. In the Administrator perspective, select **Workloads → Config Maps**.
 2. At the top right side of the page, select **Create Config Map**.
 3. Enter the contents of your config map.
 4. Select **Create**.
- To create a config map as a developer:
 1. In the Developer perspective, select **Config Maps**.
 2. At the top right side of the page, select **Create Config Map**.
 3. Enter the contents of your config map.
 4. Select **Create**.

2.10.3. Creating a config map by using the CLI

You can use the following command to create a config map from directories, specific files, or literal values.

Procedure

- Create a config map:

```
$ oc create configmap <configmap_name> [options]
```

2.10.3.1. Creating a config map from a directory

You can create a config map from a directory by using the **--from-file** flag. This method allows you to use multiple files within a directory to create a config map.

Each file in the directory is used to populate a key in the config map, where the name of the key is the file name, and the value of the key is the content of the file.

For example, the following command creates a config map with the contents of the **example-files** directory:

```
$ oc create configmap game-config --from-file=example-files/
```

View the keys in the config map:

```
$ oc describe configmaps game-config
```

Example output

```
Name:      game-config
Namespace:  default
Labels:     <none>
Annotations: <none>

Data

game.properties: 158 bytes
ui.properties:   83 bytes
```

You can see that the two keys in the map are created from the file names in the directory specified in the command. The content of those keys might be large, so the output of **oc describe** only shows the names of the keys and their sizes.

Prerequisite

- You must have a directory with files that contain the data you want to populate a config map with.

The following procedure uses these example files: **game.properties** and **ui.properties**:

```
$ cat example-files/game.properties
```

Example output

```

enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UDDLRRLRBABAS
secret.code.allowed=true
secret.code.lives=30

```

```
$ cat example-files/ui.properties
```

Example output

```

color.good=purple
color.bad=yellow
allow.textmode=true
how.nice.to.look=fairlyNice

```

Procedure

- Create a config map holding the content of each file in this directory by entering the following command:

```
$ oc create configmap game-config \
  --from-file=example-files/
```

Verification

- Enter the **oc get** command for the object with the **-o** option to see the values of the keys:

```
$ oc get configmaps game-config -o yaml
```

Example output

```

apiVersion: v1
data:
  game.properties: |-
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UDDLRRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:34:05Z
  name: game-config
  namespace: default

```

```
resourceVersion: "407"
selflink: /api/v1/namespaces/default/configmaps/game-config
uid: 30944725-d66e-11e5-8cd0-68f728db1985
```

2.10.3.2. Creating a config map from a file

You can create a config map from a file by using the **--from-file** flag. You can pass the **--from-file** option multiple times to the CLI.

You can also specify the key to set in a config map for content imported from a file by passing a **key=value** expression to the **--from-file** option. For example:

```
$ oc create configmap game-config-3 --from-file=game-special-key=example-files/game.properties
```



NOTE

If you create a config map from a file, you can include files containing non-UTF8 data that are placed in this field without corrupting the non-UTF8 data. OpenShift Container Platform detects binary files and transparently encodes the file as **MIME**. On the server, the **MIME** payload is decoded and stored without corrupting the data.

Prerequisite

- You must have a directory with files that contain the data you want to populate a config map with.

The following procedure uses these example files: **game.properties** and **ui.properties**:

```
$ cat example-files/game.properties
```

Example output

```
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UUDDLRLRBABAS
secret.code.allowed=true
secret.code.lives=30
```

```
$ cat example-files/ui.properties
```

Example output

```
color.good=purple
color.bad=yellow
allow.textmode=true
how.nice.to.look=fairlyNice
```

Procedure

- Create a config map by specifying a specific file:


```
$ oc create configmap game-config-2 \
  --from-file=example-files/game.properties \
  --from-file=example-files/ui.properties
```

- Create a config map by specifying a key-value pair:

```
$ oc create configmap game-config-3 \
  --from-file=game-special-key=example-files/game.properties
```

Verification

- Enter the **oc get** command for the object with the **-o** option to see the values of the keys from the file:

```
$ oc get configmaps game-config-2 -o yaml
```

Example output

```
apiVersion: v1
data:
  game.properties: |-
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:52:05Z
  name: game-config-2
  namespace: default
  resourceVersion: "516"
  selflink: /api/v1/namespaces/default/configmaps/game-config-2
  uid: b4952dc3-d670-11e5-8cd0-68f728db1985
```

- Enter the **oc get** command for the object with the **-o** option to see the values of the keys from the key-value pair:

```
$ oc get configmaps game-config-3 -o yaml
```

Example output

```
apiVersion: v1
data:
  game-special-key: |- 1
    enemies=aliens
```

```

    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:54:22Z
  name: game-config-3
  namespace: default
  resourceVersion: "530"
  selflink: /api/v1/namespaces/default/configmaps/game-config-3
  uid: 05f8da22-d671-11e5-8cd0-68f728db1985

```

- 1 This is the key that you set in the preceding step.

2.10.3.3. Creating a config map from literal values

You can supply literal values for a config map.

The **--from-literal** option takes a **key=value** syntax, which allows literal values to be supplied directly on the command line.

Procedure

- Create a config map by specifying a literal value:

```

$ oc create configmap special-config \
  --from-literal=special.how=very \
  --from-literal=special.type=charm

```

Verification

- Enter the **oc get** command for the object with the **-o** option to see the values of the keys:

```

$ oc get configmaps special-config -o yaml

```

Example output

```

apiVersion: v1
data:
  special.how: very
  special.type: charm
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: special-config
  namespace: default
  resourceVersion: "651"
  selflink: /api/v1/namespaces/default/configmaps/special-config
  uid: dadce046-d673-11e5-8cd0-68f728db1985

```

2.10.4. Use cases: Consuming config maps in pods

The following sections describe some uses cases when consuming **ConfigMap** objects in pods.

2.10.4.1. Populating environment variables in containers by using config maps

You can use config maps to populate individual environment variables in containers or to populate environment variables in containers from all keys that form valid environment variable names.

As an example, consider the following config map:

ConfigMap with two environment variables

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config ❶
  namespace: default ❷
data:
  special.how: very ❸
  special.type: charm ❹
```

- ❶ Name of the config map.
- ❷ The project in which the config map resides. Config maps can only be referenced by pods in the same project.
- ❸ ❹ Environment variables to inject.

ConfigMap with one environment variable

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: env-config ❶
  namespace: default
data:
  log_level: INFO ❷
```

- ❶ Name of the config map.
- ❷ Environment variable to inject.

Procedure

- You can consume the keys of this **ConfigMap** in a pod using **configMapKeyRef** sections.

Sample Pod specification configured to inject specific environment variables

```
apiVersion: v1
kind: Pod
```

```

metadata:
  name: dapi-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env: 1
        - name: SPECIAL_LEVEL_KEY 2
          valueFrom:
            configMapKeyRef:
              name: special-config 3
              key: special.how 4
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config 5
              key: special.type 6
              optional: true 7
      envFrom: 8
        - configMapRef:
            name: env-config 9
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
      restartPolicy: Never

```

1 Stanza to pull the specified environment variables from a **ConfigMap**.

2 Name of a pod environment variable that you are injecting a key's value into.

3 **5** Name of the **ConfigMap** to pull specific environment variables from.

4 **6** Environment variable to pull from the **ConfigMap**.

7 Makes the environment variable optional. As optional, the pod will be started even if the specified **ConfigMap** and keys do not exist.

8 Stanza to pull all environment variables from a **ConfigMap**.

9 Name of the **ConfigMap** to pull all environment variables from.

When this pod is run, the pod logs will include the following output:

```

SPECIAL_LEVEL_KEY=very
log_level=INFO

```

**NOTE**

SPECIAL_TYPE_KEY=charm is not listed in the example output because **optional: true** is set.

2.10.4.2. Setting command-line arguments for container commands with config maps

You can use a config map to set the value of the commands or arguments in a container by using the Kubernetes substitution syntax **\$(VAR_NAME)**.

As an example, consider the following config map:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm
```

Procedure

- To inject values into a command in a container, you must consume the keys you want to use as environment variables. Then you can refer to them in a container's command using the **\$(VAR_NAME)** syntax.

Sample pod specification configured to inject specific environment variables

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "echo $(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.how
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.type
      securityContext:
```

1

```

allowPrivilegeEscalation: false
capabilities:
  drop: [ALL]
restartPolicy: Never

```

- 1 Inject the values into a command in a container using the keys you want to use as environment variables.

When this pod is run, the output from the echo command run in the test-container container is as follows:

```

very charm

```

2.10.4.3. Injecting content into a volume by using config maps

You can inject content into a volume by using config maps.

Example ConfigMap custom resource (CR)

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm

```

Procedure

You have a couple different options for injecting content into a volume by using config maps.

- The most basic way to inject content into a volume by using a config map is to populate the volume with files where the key is the file name and the content of the file is the value of the key:

```

apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "cat", "/etc/config/special.how" ]
      volumeMounts:
        - name: config-volume
          mountPath: /etc/config
      securityContext:
        allowPrivilegeEscalation: false

```

```

    capabilities:
      drop: [ALL]
  volumes:
  - name: config-volume
    configMap:
      name: special-config 1
  restartPolicy: Never

```

- 1** File containing key.

When this pod is run, the output of the cat command will be:

```
very
```

- You can also control the paths within the volume where config map keys are projected:

```

apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: test-container
    image: gcr.io/google_containers/busybox
    command: [ "/bin/sh", "-c", "cat", "/etc/config/path/to/special-key" ]
    volumeMounts:
    - name: config-volume
      mountPath: /etc/config
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
    volumes:
    - name: config-volume
      configMap:
        name: special-config
        items:
        - key: special.how
          path: path/to/special-key 1
    restartPolicy: Never

```

- 1** Path to config map key.

When this pod is run, the output of the cat command will be:

```
very
```

2.11. MOUNTING AN OCI IMAGE INTO A POD

You can mount an Open Container Initiative (OCI)-compliant container image directly into a pod, making the files within the image accessible to the containers without the need to include them in the base image, which allows you to host the data in OCI-compliant registries.

2.11.1. Understanding image volumes

You can use an *image volume* to mount an Open Container Initiative (OCI)-compliant container image directly into a pod, making the files within the image accessible to the containers without the need to include them in the base image. This means you can host the data in an OCI-compliant registry.

By using an image volume in a pod, you can take advantage of the OCI image and distribution specification standards to accomplish several tasks including the following use cases:

- You can share configuration files among multiple containers in a pod without needing to include the file in the base image, which minimizes security risks and image size.
- In an artificial intelligence environment, you can use image volumes to mount large language model weights or machine learning model weights in a pod alongside a model-server. You can efficiently serve model weights this way without including them in the model-server container image. Therefore, you can separate the model specifications and content from the executables that process them.
- You can use a public image for a malware scanner and mount it in a volume of private malware signatures, so that you can load those signatures without incorporating the image into a base image, which might not be allowed by the copyright on the public image.

To mount an image volume, include a path to the image in your pod spec with an optional pull policy as described in *Adding an image volume to a pod*.

2.11.2. Adding an image volume to a pod

To mount an Open Container Initiative (OCI)-compliant container image, use the **volume** parameter to include a path to the image in your pod spec with an optional pull policy. You can create the pod directly or use a controlling object, such as a deployment or replica set.

Procedure

1. Create a YAML file similar to the following.

```
apiVersion: v1
kind: Pod
metadata:
  name: image-volume
spec:
  containers:
  - name: shell
    command: ["sleep", "infinity"]
    image: debian
    volumeMounts:
    - name: volume
      mountPath: /volume
  volumes:
  - name: volume
```



```
image: ❶
reference: quay.io/crio/image:v2 ❷
pullPolicy: Always ❸
```

- ❶ Specifies an OCI container image that is available on the host machine.
- ❷ Specifies the path to the image.
- ❸ Specifies a pull policy, one of the following options:
 - If **Always**, the kubelet always attempts to pull the image. If the pull fails, the kubelet sets the pod to **Failed**.
 - If **Never**, the kubelet never pulls the image and only uses a local image. The pod becomes **Failed** if any layers of the image are not present locally, or if the manifest for that image is not already cached.
 - If **IfNotPresent** the kubelet pulls the image if it is not present. The pod becomes **Failed** if the image is not present and the pull fails. This is the default.

2. Create the pod by running the following command:

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

Verification

- Examine the pod to view detailed information about the image pull and mount by using a command similar to the following:

```
$ oc describe pod <pod_name>
```

Example output

```
Name:      image-volume
Namespace: default
# ...
Volumes:
  volume: ❶
    Type: Image
    Reference: quay.io/crio/image:v2
    PullPolicy: IfNotPresent
# ...
Events:
  Type    Reason      Age      From      Message
  ----    -
  Normal  Pulling     46s      kubelet    Pulling image "quay.io/crio/image:v2"
  Normal  Pulled      44s      kubelet    Successfully pulled image
"quay.io/crio/image:v2" in 2.261s (2.261s including waiting). Image size: 6707 bytes. ❷
# ...
```

- ❶ Indicates that the image was mounted to the pod.

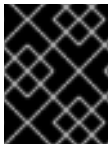
2 Indicates that the image was successfully pulled.

2.12. USING DEVICE PLUGINS TO ACCESS EXTERNAL RESOURCES WITH PODS

Device plugins allow you to use a particular device type (GPU, InfiniBand, or other similar computing resources that require vendor-specific initialization and setup) in your OpenShift Container Platform pod without needing to write custom code.

2.12.1. 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) {}
}
```

2.12.1.1. 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](https://github.com/kubernetes/kubernetes/blob/master/vendor/k8s.io/kubernetes/pkg/kubelet/cm/deviceplugin/device_plugin_stub.go).

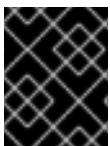
2.12.1.2. Methods for deploying a device plugin

- 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.

2.12.2. Understanding the Device Manager

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

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



IMPORTANT

OpenShift Container Platform supports the device plugin API, but the device plugin Containers are supported by individual vendors.

Device Manager advertises devices as **Extended Resources**. User pods can consume devices, advertised by Device Manager, using the same **Limit/Request** mechanism, which is used for requesting any other **Extended Resource**.

Upon start, the device plugin registers itself with Device Manager invoking **Register** on the `/var/lib/kubelet/device-plugins/kubelet.sock` and starts a gRPC service at `/var/lib/kubelet/device-plugins/<plugin>.sock` for serving Device Manager requests.

Device Manager, while processing a new registration request, invokes **ListAndWatch** remote procedure call (RPC) at the device plugin service. In response, Device Manager gets a list of **Device** objects from the plugin over a gRPC stream. Device Manager will keep watching on the stream for new updates from the plugin. On the plugin side, the plugin will also keep the stream open and whenever there is a change in the state of any of the devices, a new device list is sent to the Device Manager over the same streaming connection.

While handling a new pod admission request, Kubelet passes requested **Extended Resources** to the Device Manager for device allocation. Device Manager checks in its database to verify if a corresponding plugin exists or not. If the plugin exists and there are free allocatable devices as well as per local cache,

Allocate RPC is invoked at that particular device plugin.

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

2.12.3. Enabling Device Manager

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

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

1. Obtain the label associated with the static **MachineConfigPool** CRD for the type of node you want to configure by entering the following command. Perform one of the following steps:
 - a. View the machine config:

```
# oc describe machineconfig <name>
```

For example:

```
# oc describe machineconfig 00-worker
```

Example output

```
Name:      00-worker
Namespace:
Labels:    machineconfiguration.openshift.io/role=worker 1
```

1 Label required for the Device Manager.

Procedure

1. Create a custom resource (CR) for your configuration change.

Sample configuration for a Device Manager CR

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: devicemgr 1
spec:
  machineConfigPoolSelector:
    matchLabels:
      machineconfiguration.openshift.io: devicemgr 2
  kubeletConfig:
    feature-gates:
      - DevicePlugins=true 3
```

1 Assign a name to CR.

- 2 Enter the label from the Machine Config Pool.
- 3 Set **DevicePlugins** to 'true`.

2. Create the Device Manager:

```
$ oc create -f devicemgr.yaml
```

Example output

```
kubeletconfig.machineconfiguration.openshift.io/devicemgr created
```

3. Ensure that Device Manager was actually enabled by confirming that `/var/lib/kubelet/device-plugins/kubelet.sock` is created on the node. This is the UNIX domain socket on which the Device Manager gRPC server listens for new plugin registrations. This sock file is created when the Kubelet is started only if Device Manager is enabled.

2.13. INCLUDING POD PRIORITY IN POD SCHEDULING DECISIONS

You can enable pod priority and preemption in your cluster. Pod priority indicates the importance of a pod relative to other pods and queues the pods based on that priority. pod preemption allows the cluster to evict, or preempt, lower-priority pods so that higher-priority pods can be scheduled if there is no available space on a suitable node pod priority also affects the scheduling order of pods and out-of-resource eviction ordering on the node.

To use priority and preemption, you create priority classes that define the relative weight of your pods. Then, reference a priority class in the pod specification to apply that weight for scheduling.

2.13.1. Understanding pod priority

When you use the Pod Priority and Preemption feature, the scheduler orders pending pods by their priority, and a pending pod is placed ahead of other pending pods with lower priority in the scheduling queue. As a result, the higher priority pod might be scheduled sooner than pods with lower priority if its scheduling requirements are met. If a pod cannot be scheduled, scheduler continues to schedule other lower priority pods.

2.13.1.1. Pod priority classes

You can assign pods a priority class, which is a non-namespaced object that defines a mapping from a name to the integer value of the priority. The higher the value, the higher the priority.

A priority class object can take any 32-bit integer value smaller than or equal to 1000000000 (one billion). Reserve numbers larger than or equal to one billion for critical pods that must not be preempted or evicted. By default, OpenShift Container Platform has two reserved priority classes for critical system pods to have guaranteed scheduling.

```
$ oc get priorityclasses
```

Example output

NAME	VALUE	GLOBAL-DEFAULT	AGE
system-node-critical	2000001000	false	72m

system-cluster-critical	2000000000	false	72m
openshift-user-critical	1000000000	false	3d13h
cluster-logging	1000000	false	29s

- **system-node-critical** - This priority class has a value of 2000001000 and is used for all pods that should never be evicted from a node. Examples of pods that have this priority class are **ovnkube-node**, and so forth. A number of critical components include the **system-node-critical** priority class by default, for example:
 - master-api
 - master-controller
 - master-etcd
 - ovn-kubernetes
 - sync
- **system-cluster-critical** - This priority class has a value of 2000000000 (two billion) and is used with pods that are important for the cluster. Pods with this priority class can be evicted from a node in certain circumstances. For example, pods configured with the **system-node-critical** priority class can take priority. However, this priority class does ensure guaranteed scheduling. Examples of pods that can have this priority class are fluentd, add-on components like descheduler, and so forth. A number of critical components include the **system-cluster-critical** priority class by default, for example:
 - fluentd
 - metrics-server
 - descheduler
- **openshift-user-critical** - You can use the **priorityClassName** field with important pods that cannot bind their resource consumption and do not have predictable resource consumption behavior. Prometheus pods under the **openshift-monitoring** and **openshift-user-workload-monitoring** namespaces use the **openshift-user-critical priorityClassName**. Monitoring workloads use **system-critical** as their first **priorityClass**, but this causes problems when monitoring uses excessive memory and the nodes cannot evict them. As a result, monitoring drops priority to give the scheduler flexibility, moving heavy workloads around to keep critical nodes operating.
- **cluster-logging** - This priority is used by Fluentd to make sure Fluentd pods are scheduled to nodes over other apps.

2.13.1.2. Pod priority names

After you have one or more priority classes, you can create pods that specify a priority class name in a **Pod** spec. The priority admission controller uses the priority class name field to populate the integer value of the priority. If the named priority class is not found, the pod is rejected.

2.13.2. Understanding pod preemption

When a developer creates a pod, the pod goes into a queue. If the developer configured the pod for pod priority or preemption, the scheduler picks a pod from the queue and tries to schedule the pod on a node. If the scheduler cannot find space on an appropriate node that satisfies all the specified

requirements of the pod, preemption logic is triggered for the pending pod.

When the scheduler preempts one or more pods on a node, the **nominatedNodeName** field of higher-priority **Pod** spec is set to the name of the node, along with the **nodeName** field. The scheduler uses the **nominatedNodeName** field to keep track of the resources reserved for pods and also provides information to the user about preemptions in the clusters.

After the scheduler preempts a lower-priority pod, the scheduler honors the graceful termination period of the pod. If another node becomes available while scheduler is waiting for the lower-priority pod to terminate, the scheduler can schedule the higher-priority pod on that node. As a result, the **nominatedNodeName** field and **nodeName** field of the **Pod** spec might be different.

Also, if the scheduler preempts pods on a node and is waiting for termination, and a pod with a higher-priority pod than the pending pod needs to be scheduled, the scheduler can schedule the higher-priority pod instead. In such a case, the scheduler clears the **nominatedNodeName** of the pending pod, making the pod eligible for another node.

Preemption does not necessarily remove all lower-priority pods from a node. The scheduler can schedule a pending pod by removing a portion of the lower-priority pods.

The scheduler considers a node for pod preemption only if the pending pod can be scheduled on the node.

2.13.2.1. Non-preempting priority classes

Pods with the preemption policy set to **Never** are placed in the scheduling queue ahead of lower-priority pods, but they cannot preempt other pods. A non-preempting pod waiting to be scheduled stays in the scheduling queue until sufficient resources are free and it can be scheduled. Non-preempting pods, like other pods, are subject to scheduler back-off. This means that if the scheduler tries unsuccessfully to schedule these pods, they are retried with lower frequency, allowing other pods with lower priority to be scheduled before them.

Non-preempting pods can still be preempted by other, high-priority pods.

2.13.2.2. Pod preemption and other scheduler settings

If you enable pod priority and preemption, consider your other scheduler settings:

Pod priority and pod disruption budget

A pod disruption budget specifies the minimum number or percentage of replicas that must be up at a time. If you specify pod disruption budgets, OpenShift Container Platform respects them when preempting pods at a best effort level. The scheduler attempts to preempt pods without violating the pod disruption budget. If no such pods are found, lower-priority pods might be preempted despite their pod disruption budget requirements.

Pod priority and pod affinity

Pod affinity requires a new pod to be scheduled on the same node as other pods with the same label.

If a pending pod has inter-pod affinity with one or more of the lower-priority pods on a node, the scheduler cannot preempt the lower-priority pods without violating the affinity requirements. In this case, the scheduler looks for another node to schedule the pending pod. However, there is no guarantee that the scheduler can find an appropriate node and pending pod might not be scheduled.

To prevent this situation, carefully configure pod affinity with equal-priority pods.

2.13.2.3. Graceful termination of preempted pods

When preempting a pod, the scheduler waits for the pod graceful termination period to expire, allowing the pod to finish working and exit. If the pod does not exit after the period, the scheduler kills the pod. This graceful termination period creates a time gap between the point that the scheduler preempts the pod and the time when the pending pod can be scheduled on the node.

To minimize this gap, configure a small graceful termination period for lower-priority pods.

2.13.3. Configuring priority and preemption

You apply pod priority and preemption by creating a priority class object and associating pods to the priority by using the **priorityClassName** in your pod specs.



NOTE

You cannot add a priority class directly to an existing scheduled pod.

Procedure

To configure your cluster to use priority and preemption:

1. Create one or more priority classes:
 - a. Create a YAML file similar to the following:

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority 1
  value: 1000000 2
  preemptionPolicy: PreemptLowerPriority 3
  globalDefault: false 4
  description: "This priority class should be used for XYZ service pods only." 5
```

- 1** The name of the priority class object.
- 2** The priority value of the object.
- 3** Optional. Specifies whether this priority class is preempting or non-preempting. The preemption policy defaults to **PreemptLowerPriority**, which allows pods of that priority class to preempt lower-priority pods. If the preemption policy is set to **Never**, pods in that priority class are non-preempting.
- 4** Optional. Specifies whether this priority class should be used for pods without a priority class name specified. This field is **false** by default. Only one priority class with **globalDefault** set to **true** can exist in the cluster. If there is no priority class with **globalDefault:true**, the priority of pods with no priority class name is zero. Adding a priority class with **globalDefault:true** affects only pods created after the priority class is added and does not change the priorities of existing pods.
- 5** Optional. Describes which pods developers should use with this priority class. Enter an arbitrary text string.

- b. Create the priority class:

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

2. Create a pod spec to include the name of a priority class:

- a. Create a YAML file similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
    securityContext:
      allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
    priorityClassName: high-priority 1
```

- 1 Specify the priority class to use with this pod.

- b. Create the pod:

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

You can add the priority name directly to the pod configuration or to a pod template.

2.14. PLACING PODS ON SPECIFIC NODES USING NODE SELECTORS

A *node selector* specifies a map of key-value pairs. The rules are defined using custom labels on nodes and selectors specified in pods.

For the pod to be eligible to run on a node, the pod must have the indicated key-value pairs as the label on the node.

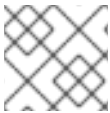
If you are using node affinity and node selectors in the same pod configuration, see the important considerations below.

2.14.1. Using node selectors to control pod placement

You can use node selectors on pods and labels on nodes to control where the pod is scheduled. With node selectors, OpenShift Container Platform schedules the pods on nodes that contain matching labels.

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.

To add node selectors to an existing pod, add a node selector to the controlling object for that pod, such as a **ReplicaSet** object, **DaemonSet** object, **StatefulSet** object, **Deployment** object, or **DeploymentConfig** object. Any existing pods under that controlling object are recreated on a node with a matching label. If you are creating a new pod, you can add the node selector directly to the pod spec. If the pod does not have a controlling object, you must delete the pod, edit the pod spec, and recreate the pod.



NOTE

You cannot add a node selector directly to an existing scheduled pod.

Prerequisites

To add a node selector to existing pods, determine the controlling object for that pod. For example, the **router-default-66d5cf9464-m2g75** pod is controlled by the **router-default-66d5cf9464** replica set:

```
$ oc describe pod router-default-66d5cf9464-7pwkc
```

Example output

```
kind: Pod
apiVersion: v1
metadata:
# ...
Name:      router-default-66d5cf9464-7pwkc
Namespace: openshift-ingress
# ...
Controlled By:  ReplicaSet/router-default-66d5cf9464
# ...
```

The web console lists the controlling object under **ownerReferences** in the pod YAML:

```
apiVersion: v1
kind: Pod
metadata:
  name: router-default-66d5cf9464-7pwkc
# ...
ownerReferences:
- apiVersion: apps/v1
  kind: ReplicaSet
  name: router-default-66d5cf9464
  uid: d81dd094-da26-11e9-a48a-128e7edf0312
  controller: true
  blockOwnerDeletion: true
# ...
```

Procedure

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

- Use a **MachineSet** object 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
```

For example:

```
$ oc patch MachineSet abc612-msrtw-worker-us-east-1c --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: xf2bd-infra-us-east-2a
  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
    # ...

```

- 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 ip-10-0-142-25.ec2.internal 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: hello-node-6fbccf8d9
  labels:
    type: "user-node"
    region: "east"
# ...

```

- b. Verify that the labels are added to the node:

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

Example output

```

NAME                                STATUS  ROLES  AGE  VERSION
ip-10-0-142-25.ec2.internal  Ready  worker  17m  v1.33.4

```

2. Add the matching node selector to a pod:

- To add a node selector to existing and future pods, add a node selector to the controlling object for the pods:

Example ReplicaSet object with labels

```

kind: ReplicaSet
apiVersion: apps/v1
metadata:
  name: hello-node-6fbccf8d9
# ...
spec:
# ...
  template:
    metadata:

```

```

creationTimestamp: null
labels:
  ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
  pod-template-hash: 66d5cf9464
spec:
  nodeSelector:
    kubernetes.io/os: linux
    node-role.kubernetes.io/worker: "
  type: user-node 1
# ...

```

1 Add the node selector.

- To add a node selector to a specific, new pod, add the selector to the **Pod** object directly:

Example Pod object with a node selector

```

apiVersion: v1
kind: Pod
metadata:
  name: hello-node-6fbccf8d9
# ...
spec:
  nodeSelector:
    region: east
    type: user-node
# ...

```

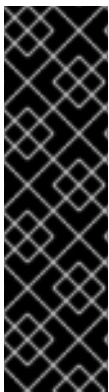


NOTE

You cannot add a node selector directly to an existing scheduled pod.

2.15. ALLOCATING GPUS TO PODS

Attribute-Based GPU Allocation enables fine-tuned control over graphics processing unit (GPU) resource allocation in OpenShift Container Platform, allowing pods to request GPUs based on specific device attributes, including product name, GPU memory capacity, compute capability, vendor name and driver version. These attributes are exposed by a third-party Dynamic Resource Allocation (DRA) driver.



IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

2.15.1. About allocating GPUs to workloads

Attribute-Based GPU Allocation enables pods to request graphics processing units (GPU) based on specific device attributes. This ensures that each pod receives the exact GPU specifications it requires.

Attribute-based resource allocation requires that you install a Dynamic Resource Allocation (DRA) driver. A DRA driver is a third-party application that runs on each node in your cluster to interface with the hardware of that node.

The DRA driver advertises several GPU device attributes that OpenShift Container Platform can use for precise GPU selection, including the following attributes:

Product Name

Pods can request an exact GPU model based on performance requirements or compatibility with applications. This ensures that workloads leverage the best-suited hardware for their tasks.

GPU Memory Capacity

Pods can request GPUs with a minimum or maximum memory capacity, such as 8 GB, 16 GB, or 40 GB. This is helpful with memory-intensive workloads such as large AI model training or data processing. This attribute enables applications to allocate GPUs that meet memory needs without overcommitting or underutilizing resources.

Compute Capability

Pods can request GPUs based on the compute capabilities of the GPU, such as the CUDA versions supported. Pods can target GPUs that are compatible with the application's framework and leverage optimized processing capabilities.

Power and Thermal Profiles

Pods can request GPUs based on power usage or thermal characteristics, enabling power-sensitive or temperature-sensitive applications to operate efficiently. This is particularly useful in high-density environments where energy or cooling constraints are factors.

Device ID and Vendor ID

Pods can request GPUs based on the GPU's hardware specifics, which allows applications that require specific vendors or device types to make targeted requests.

Driver Version

Pods can request GPUs that run a specific driver version, ensuring compatibility with application dependencies and maximizing GPU feature access.

2.15.2. About GPU allocation objects

Attribute-Based GPU Allocation uses the following objects to provide the core graphics processing unit (GPU) allocation functionality. All of these API kinds are included in the **resource.k8s.io/v1beta2** API group.

Device class

A device class is a category of devices that pods can claim and how to select specific device attributes in claims. Some device drivers contain their own device class. Alternatively, an administrator can create device classes. A device class contains a device selector, which is a [common expression language \(CEL\)](#) expression that must evaluate to true if a device satisfies the request. The following example **DeviceClass** object selects any device that is managed by the **driver.example.com** device driver:

Example device class object

```
apiVersion: resource.k8s.io/v1beta1
kind: DeviceClass
```

```

metadata:
  name: example-device-class
spec:
  selectors:
  - cel:
      expression: |-
        device.driver == "driver.example.com"

```

Resource slice

The Dynamic Resource Allocation (DRA) driver on each node creates and manages *resource slices* in the cluster. A resource slice represents one or more GPU resources that are attached to nodes. When a resource claim is created and used in a pod, OpenShift Container Platform uses the resource slices to find nodes that have access to the requested resources. After finding an eligible resource slice for the resource claim, the OpenShift Container Platform scheduler updates the resource claim with the allocation details, allocates resources to the resource claim, and schedules the pod onto a node that can access the resources.

Resource claim template

Cluster administrators and operators can create a *resource claim template* to request a GPU from a specific device class. Resource claim templates provide pods with access to separate, similar resources. OpenShift Container Platform uses a resource claim template to generate a resource claim for the pod. Each resource claim that OpenShift Container Platform generates from the template is bound to a specific pod. When the pod terminates, OpenShift Container Platform deletes the corresponding resource claim.

The following example resource claim template requests devices in the **example-device-class** device class.

Example resource claim template object

```

apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  namespace: gpu-test1
  name: gpu-claim-template
spec:
  # ...
  spec:
    devices:
      requests:
      - name: gpu
        deviceClassName: example-device-class

```

Resource claim

Admins and operators can create a *resource claim* to request a GPU from a specific device class. A resource claim differs from a resource claim template by allowing you to share GPUs with multiple pods. Also, resource claims are not deleted when a requesting pod is terminated.

The following example resource claim template uses CEL expressions to request specific devices in the **example-device-class** device class that are of a specific size.

Example resource claim object

```

apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaim
metadata:

```

```

namespace: gpu-claim
name: gpu-devices
spec:
  devices:
    requests:
      - name: 1g-5gb
        deviceClassName: example-device-class
        selectors:
          - cel:
              expression: "device.attributes['driver.example.com'].profile == '1g.5gb'"
      - name: 1g-5gb-2
        deviceClassName: example-device-class
        selectors:
          - cel:
              expression: "device.attributes['driver.example.com'].profile == '1g.5gb'"
      - name: 2g-10gb
        deviceClassName: example-device-class
        selectors:
          - cel:
              expression: "device.attributes['driver.example.com'].profile == '2g.10gb'"
      - name: 3g-20gb
        deviceClassName: example-device-class
        selectors:
          - cel:
              expression: "device.attributes['driver.example.com'].profile == '3g.20gb'"

```

For more information on configuring resource claims, resource claim templates, see ["Dynamic Resource Allocation"](#) (Kubernetes documentation).

For information on adding resource claims to pods, see ["Adding resource claims to pods"](#).

Next steps

- [Adding resource claims to pods](#)

2.15.3. Adding resource claims to pods

Attribute-Based GPU Allocation uses resource claims and resource claim templates to allow you to request specific graphics processing units (GPU) for the containers in your pods. Resource claims can be used with multiple containers, but resource claim templates can be used with only one container. For more information, see ["About configuring device allocation by using device attributes"](#) in the *Additional Resources* section.

The example in the following procedure creates a resource claim template to assign a specific GPU to **container0** and a resource claim to share a GPU between **container1** and **container2**.

Prerequisites

- A Dynamic Resource Allocation (DRA) driver is installed. For more information on DRA, see ["Dynamic Resource Allocation"](#) (Kubernetes documentation).
- A resource slice has been created.
- A resource claim and/or resource claim template has been created.

- You enabled the required Technology Preview features for your cluster by editing the **FeatureGate** CR named **cluster**:

Example FeatureGate CR

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

- 1** Enables the required features.



WARNING

Enabling the **TechPreviewNoUpgrade** feature set on your cluster cannot be undone and prevents minor version updates. This feature set allows you to enable these Technology Preview features on test clusters, where you can fully test them. Do not enable this feature set on production clusters.

Procedure

1. Create a pod by creating a YAML file similar to the following:

Example pod that is requesting resources

```
apiVersion: v1
kind: Pod
metadata:
  namespace: gpu-allocate
  name: pod1
  labels:
    app: pod
spec:
  restartPolicy: Never
  containers:
    - name: container0
      image: ubuntu:24.04
      command: ["sleep", "9999"]
      resources:
        claims: 1
        - name: gpu-claim-template
    - name: container1
      image: ubuntu:24.04
      command: ["sleep", "9999"]
      resources:
        claims:
          - name: gpu-claim
```

```

- name: container2
  image: ubuntu:24.04
  command: ["sleep", "9999"]
  resources:
    claims:
      - name: gpu-claim
  resourceClaims: 2
  - name: gpu-claim-template
    resourceClaimTemplateName: example-resource-claim-template
  - name: gpu-claim
    resourceClaimName: example-resource-claim

```

- 1 Specifies one or more resource claims to use with this container.
- 2 Specifies the resource claims that are required for the containers to start. Include an arbitrary name for the resource claim request and the resource claim and/or resource claim template.

2. Create the CRD object:

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

For more information on configuring pod resource requests, see ["Dynamic Resource Allocation"](#) (Kubernetes documentation).

Additional resources

- [About configuring device allocation by using device attributes](#)

2.16. RUN ONCE DURATION OVERRIDE OPERATOR

2.16.1. Run Once Duration Override Operator overview

You can use the Run Once Duration Override Operator to specify a maximum time limit that run-once pods can be active for.

2.16.1.1. About the Run Once Duration Override Operator

OpenShift Container Platform relies on run-once pods to perform tasks such as deploying a pod or performing a build. Run-once pods are pods that have a **RestartPolicy** of **Never** or **OnFailure**.

Cluster administrators can use the Run Once Duration Override Operator to force a limit on the time that those run-once pods can be active. After the time limit expires, the cluster will try to actively terminate those pods. The main reason to have such a limit is to prevent tasks such as builds to run for an excessive amount of time.

To apply the run-once duration override from the Run Once Duration Override Operator to run-once pods, you must enable it on each applicable namespace.

If both the run-once pod and the Run Once Duration Override Operator have their **activeDeadlineSeconds** value set, the lower of the two values is used.

**NOTE**

You cannot install the Run Once Duration Override Operator on clusters managed by the HyperShift Operator.

2.16.2. Run Once Duration Override Operator release notes

Cluster administrators can use the Run Once Duration Override Operator to force a limit on the time that run-once pods can be active. After the time limit expires, the cluster tries to terminate the run-once pods. The main reason to have such a limit is to prevent tasks such as builds to run for an excessive amount of time.

To apply the run-once duration override from the Run Once Duration Override Operator to run-once pods, you must enable it on each applicable namespace.

These release notes track the development of the Run Once Duration Override Operator for OpenShift Container Platform.

For an overview of the Run Once Duration Override Operator, see [About the Run Once Duration Override Operator](#).

2.16.2.1. Run Once Duration Override Operator 1.3.1

Issued: 29 October 2025

The following advisory is available for the Run Once Duration Override Operator 1.3.1: ([RHBA-2025:19272](#))

2.16.2.1.1. New features and enhancements

- This release of the Run Once Duration Override Operator updates the Kubernetes version to 1.33.

2.16.2.1.2. Bug fixes

- This release of the Run Once Duration Override Operator addresses several Common Vulnerabilities and Exposures (CVEs).

2.16.2.2. Run Once Duration Override Operator 1.3.0

Issued: 9 July 2025

The following advisory is available for the Run Once Duration Override Operator 1.3.0: ([RHBA-2025-10725](#))

2.16.2.2.1. Bug fixes

- This release of the Run Once Duration Override Operator addresses several Common Vulnerabilities and Exposures (CVEs).

2.16.3. Overriding the active deadline for run-once pods

You can use the Run Once Duration Override Operator to specify a maximum time limit that run-once pods can be active for. By enabling the run-once duration override on a namespace, all future run-once pods created or updated in that namespace have their **activeDeadlineSeconds** field set to the value

specified by the Run Once Duration Override Operator.



NOTE

If both the run-once pod and the Run Once Duration Override Operator have their **activeDeadlineSeconds** value set, the lower of the two values is used.

2.16.3.1. Installing the Run Once Duration Override Operator

You can use the web console to install the Run Once Duration Override Operator.

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. Create the required namespace for the Run Once Duration Override Operator.
 - a. Navigate to **Administration** → **Namespaces** and click **Create Namespace**.
 - b. Enter **openshift-run-once-duration-override-operator** in the **Name** field and click **Create**.
3. Install the Run Once Duration Override Operator.
 - a. Navigate to **Ecosystem** → **Software Catalog**.
 - b. Enter **Run Once Duration Override Operator** into the filter box.
 - c. Select the **Run Once Duration Override Operator** and click **Install**.
 - d. On the **Install Operator** page:
 - i. The **Update channel** is set to **stable**, which installs the latest stable release of the Run Once Duration Override Operator.
 - ii. Select **A specific namespace on the cluster**
 - iii. Choose **openshift-run-once-duration-override-operator** from the dropdown menu under **Installed namespace**.
 - iv. Select an **Update approval** strategy.
 - The **Automatic** strategy allows Operator Lifecycle Manager (OLM) to automatically update the Operator when a new version is available.
 - The **Manual** strategy requires a user with appropriate credentials to approve the Operator update.
 - v. Click **Install**.
4. Create a **RunOnceDurationOverride** instance.

- a. From the **Ecosystem → Installed Operators** page, click **Run Once Duration Override Operator**.
- b. Select the **Run Once Duration Override** tab and click **Create RunOnceDurationOverride**.
- c. Edit the settings as necessary.
Under the **runOnceDurationOverride** section, you can update the **spec.activeDeadlineSeconds** value, if required. The predefined value is **3600** seconds, or 1 hour.
- d. Click **Create**.

Verification

1. Log in to the OpenShift CLI.
2. Verify all pods are created and running properly.

```
$ oc get pods -n openshift-run-once-duration-override-operator
```

Example output

NAME	READY	STATUS	RESTARTS	AGE
run-once-duration-override-operator-7b88c676f6-lcxgc	1/1	Running	0	7m46s
runoncedurationoverride-62blp	1/1	Running	0	41s
runoncedurationoverride-h8h8b	1/1	Running	0	41s
runoncedurationoverride-tdsqk	1/1	Running	0	41s

2.16.3.2. Enabling the run-once duration override on a namespace

To apply the run-once duration override from the Run Once Duration Override Operator to run-once pods, you must enable it on each applicable namespace.

Prerequisites

- The Run Once Duration Override Operator is installed.

Procedure

1. Log in to the OpenShift CLI.
2. Add the label to enable the run-once duration override to your namespace:

```
$ oc label namespace <namespace> \ 1
runoncedurationoverrides.admission.runoncedurationoverride.openshift.io/enabled=true
```

- 1** Specify the namespace to enable the run-once duration override on.

After you enable the run-once duration override on this namespace, future run-once pods that are created in this namespace will have their **activeDeadlineSeconds** field set to the override value from the Run Once Duration Override Operator. Existing pods in this namespace will also have their **activeDeadlineSeconds** value set when they are updated next.

Verification

1. Create a test run-once pod in the namespace that you enabled the run-once duration override on:

```
apiVersion: v1
kind: Pod
metadata:
  name: example
  namespace: <namespace>
spec:
  restartPolicy: Never
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: busybox
    securityContext:
      allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
    image: busybox:1.25
    command:
      - /bin/sh
      - -ec
      - |
        while sleep 5; do date; done
```

- 1 Replace **<namespace>** with the name of your namespace.
- 2 The **restartPolicy** must be **Never** or **OnFailure** to be a run-once pod.

2. Verify that the pod has its **activeDeadlineSeconds** field set:

```
$ oc get pods -n <namespace> -o yaml | grep activeDeadlineSeconds
```

Example output

```
activeDeadlineSeconds: 3600
```

2.16.3.3. Updating the run-once active deadline override value

You can customize the override value that the Run Once Duration Override Operator applies to run-once pods. The predefined value is **3600** seconds, or 1 hour.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have installed the Run Once Duration Override Operator.

Procedure

1. Log in to the OpenShift CLI.
2. Edit the **RunOnceDurationOverride** resource:

```
$ oc edit runoncedurationoverride cluster
```

3. Update the **activeDeadlineSeconds** field:

```
apiVersion: operator.openshift.io/v1
kind: RunOnceDurationOverride
metadata:
# ...
spec:
  runOnceDurationOverride:
    spec:
      activeDeadlineSeconds: 1800 1
# ...
```

- 1 Set the **activeDeadlineSeconds** field to the desired value, in seconds.

4. Save the file to apply the changes.

Any future run-once pods created in namespaces where the run-once duration override is enabled will have their **activeDeadlineSeconds** field set to this new value. Existing run-once pods in these namespaces will receive this new value when they are updated.

2.16.4. Uninstalling the Run Once Duration Override Operator

You can remove the Run Once Duration Override Operator from OpenShift Container Platform by uninstalling the Operator and removing its related resources.

2.16.4.1. Uninstalling the Run Once Duration Override Operator

You can use the web console to uninstall the Run Once Duration Override Operator. Uninstalling the Run Once Duration Override Operator does not unset the **activeDeadlineSeconds** field for run-once pods, but it will no longer apply the override value to future run-once pods.

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 Run Once Duration Override Operator.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Ecosystem** → **Installed Operators**.
3. Select **openshift-run-once-duration-override-operator** from the **Project** dropdown list.

4. Delete the **RunOnceDurationOverride** instance.

- a. Click **Run Once Duration Override Operator** and select the **Run Once Duration Override** tab.



- b. Click the Options menu next to the **cluster** entry and select **Delete RunOnceDurationOverride**.

- c. In the confirmation dialog, click **Delete**.

5. Uninstall the Run Once Duration Override Operator.

- a. Navigate to **Ecosystem → Installed Operators**.



- b. Click the Options menu next to the **Run Once Duration Override Operator** entry and click **Uninstall Operator**.

- c. In the confirmation dialog, click **Uninstall**.

2.16.4.2. Uninstalling Run Once Duration Override Operator resources

Optionally, after uninstalling the Run Once Duration Override Operator, you can remove its related resources from your cluster.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OpenShift Container Platform web console.
- You have uninstalled the Run Once Duration Override Operator.

Procedure

1. Log in to the OpenShift Container Platform web console.
 2. Remove CRDs that were created when the Run Once Duration Override Operator was installed:
 - a. Navigate to **Administration → CustomResourceDefinitions**.
 - b. Enter **RunOnceDurationOverride** in the **Name** field to filter the CRDs.
- 
- c. Click the Options menu next to the **RunOnceDurationOverride** CRD and select **Delete CustomResourceDefinition**.
 - d. In the confirmation dialog, click **Delete**.
3. Delete the **openshift-run-once-duration-override-operator** namespace.
 - a. Navigate to **Administration → Namespaces**.
 - b. Enter **openshift-run-once-duration-override-operator** into the filter box.

- c. Click the Options menu  next to the **openshift-run-once-duration-override-operator** entry and select **Delete Namespace**.
 - d. In the confirmation dialog, enter **openshift-run-once-duration-override-operator** and click **Delete**.
- 4. Remove the run-once duration override label from the namespaces that it was enabled on.
 - a. Navigate to **Administration → Namespaces**.
 - b. Select your namespace.
 - c. Click **Edit** next to the **Labels** field.
 - d. Remove the **runoncedurationoverrides.admission.runoncedurationoverride.openshift.io/enabled=true** label and click **Save**.

2.17. RUNNING PODS IN LINUX USER NAMESPACES

Linux user namespaces allow administrators to isolate the container user and group identifiers (UIDs and GIDs) so that a container can have a different set of permissions in the user namespace than on the host system where it is running. This allows containers to run processes with full privileges inside the user namespace, but the processes can be unprivileged for operations on the host machine.

By default, a container runs in the host user namespace. Running a container in the host user namespace can be useful when the container needs a feature that is available only in the host namespace. However, running pods in the host namespace introduces security concerns, such as the possibility of container breakouts, in which a process inside another container breaks out onto the host where the process can access or modify files on the host or in your containers.

Running containers in individual user namespaces can mitigate container breakouts and several other vulnerabilities that a compromised container can pose to other pods and the node itself.

When running a pod in an isolated user namespace, the UID/GID inside a pod container no longer matches the UID/GID on the host. In order for file system ownership to work correctly, the Linux kernel uses ID-mapped mounts, which translate user IDs between the container and the host at the virtual file system (VFS) layer.



IMPORTANT

Not all file systems currently support ID-mapped mounts, such as Network File Systems (NFS) and other network/distributed file systems. Any pod that is using an NFS-backed persistent volume from a vendor that does not support ID-mapped mounts might experience access or permission issues when running in a user namespace. This behavior is not specific to OpenShift Container Platform. It applies to all Kubernetes distributions from Kubernetes v1.33 onward.

2.17.1. Configuring Linux user namespace support

You can configure Linux user namespace by setting the **hostUsers** parameter to **false** in the pod spec, and a few other configurations, as shown in the following procedure.

Running workloads in user namespaces makes it safe to configure **RunAsAny** for Security Context Constraint (SCC) fields, such as **fsGroup**, **runAsGroup**, **runAsUser**, and **supplementalGroups**, as the UID or GID outside of the container is different from the one inside, which these fields express.

For extra security, you can use the **restricted-v3** or **nested-container** SCC, which are specifically designed for workloads in Linux user namespaces. The **userNamespaceLevel: RequirePodLevel** field in the SCC requires that the workloads run in user namespaces. For more information about SCCs, see "Managing security context constraints".

To require a specific SCC for a workload, you can add an SCC to a specific user or group by using the **oc adm policy add-scc-to-user** or **oc adm policy add-scc-to-group** command. For more information, see the "OpenShift CLI administrator command reference".

Also, you can optionally use the **procMount** parameter in a pod specification to configure the **/proc** file system in pods as **unmasked**. Setting **/proc** to **unmasked**, which is generally considered as safe, bypasses the default masking behavior of the container runtime, and should be used only with an SCC that sets **userNamespaceLevel** to **RequirePodLevel**.

Prerequisites

- Log into an OpenShift Container Platform cluster as a user configured with the **restricted-v3** or **nested-container** SCC, or as a user from a user group configured with either SCC. Alternatively, you can set the **restricted-v3** or **nested-container** SCC directly in the workload object.

Procedure

- Edit the default user ID (UID) and group ID (GID) range of the OpenShift Container Platform namespace where your pod is deployed by running the following command:

```
$ oc edit ns/<namespace_name>
```

Example namespace

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    openshift.io/description: ""
    openshift.io/display-name: ""
    openshift.io/requester: system:admin
    openshift.io/sa.scc.mcs: s0:c27,c24
    openshift.io/sa.scc.supplemental-groups: 1000/10000
    openshift.io/sa.scc.uid-range: 1000/10000
  # ...
  name: userns
  # ...
```

where:

metadata.annotations.openshift.io/sa.scc.supplemental-groups

Specifies the default GID to require in the pod spec. The range for a Linux user namespace must be **65535** or lower. The default is **1000000000/10000**.

metadata.annotations.openshift.io/sa.scc.uid-range

Specifies the default UID to require in the pod spec. The range for a Linux user namespace must be **65535** or lower. The default is **1000000000/10000**.



NOTE

The range 1000/10000 means 10,000 values starting with ID 1000, so it specifies the range of IDs from 1000 to 10,999.

2. Enable the use of Linux user namespaces by creating a workload configured to run with an appropriate SCC and the **hostUsers** parameter set to **false**.
 - a. Create a YAML file similar to the following:

Example pod specification

```
apiVersion: v1
kind: Pod
metadata:
  name: userns-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  hostUsers: false
  containers:
    name: userns-container
    image: registry.access.redhat.com/ubi9
    command: ["sleep", "1000"]
    securityContext:
      runAsUser: 1000
      runAsGroup: 1000
      runAsNonRoot: true
      procMount: Unmasked
      capabilities:
        add:
          - "SETGID"
          - "SETUID"
# ...
```

where:

spec.hostUsers

Specifies whether the pod is to be run in a user namespace. If **false**, the pod runs in a new user namespace that is created for the pod. If **true**, the pod runs in the host user namespace. The default is **true**.

spec.containers.securityContext.runAsUser

Specifies the user ID for processes that run inside of the container. This must fall in the range that you set in the **namespace** object.

spec.containers.securityContext.runAsGroup

Specifies the group ID for processes that run inside of the containers. This must fall in the range that you set in the **namespace** object.

spec.containers.securityContext.runAsNonRoot

Specifies that processes inside the container run with a user that has any UID other than 0.

spec.containers.securityContext.procMount

Specifies the type of proc mount to use for the containers. The **unmasked** value ensures that a container's **/proc** file system is mounted as read/write by the container process. The default is **Default**. This value is optional.

spec.containers.securityContext.capabilities.add

Specifies Linux capabilities to set in the user namespace. This permits privileged actions without giving full root access. Technically, setting capabilities inside of a user namespace is safer than setting them outside, as the scope of the capabilities are limited by being inside user namespace, and can generally be considered to be safe. However, giving pods capabilities like **CAP_SYS_ADMIN** to any untrusted workload could increase the potential kernel surface area that a containerized process has access to and could find exploits in. Thus, capabilities inside of a user namespace are allowed at **baseline** level in pod security admission. This value is optional.

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

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

Verification

1. Check the user and group IDs being used by the container in the pod you created. The pod is inside the Linux user namespace.
 - a. Start a shell session with the container in your pod:

```
$ oc rsh -c <container_name> pod/<pod_name>
```

Example command

```
$ oc rsh -c userns-container pod/userns-pod
```

- b. Display the user and group IDs being used inside the container:

```
sh-5.1$ id
```

Example output

```
uid=1000(1000) gid=1000(1000) groups=1000(1000) 1
```

- 1** The UID and group for the container should be the same as you set in the pod specification.

- c. Display the user ID being used in the container user namespace:

```
sh-5.1$ lsns -t user
```

Example output

```

      NS TYPE NPROCS PID USER COMMAND
4026532447 user    3  1 1000 /usr/bin/coreutils --coreutils-prog-shebang=sleep
/usr/bin/sleep 1000 ❶

```

- ❶ The UID for the process should be the same as you set in the pod spec.

- d. Exit the container shell session by using the following command:

```
sh-5.1$ exit
```

2. Check the UID being used by the node. The node is outside of the Linux user namespace. This user ID should be different from the UID being used in the container.

- a. Start a debug session for that node:

```
$ oc debug node/ci-ln-z5vppzb-72292-8zp2b-worker-c-q8sh9
```

Example command

```
$ oc debug node/ci-ln-z5vppzb-72292-8zp2b-worker-c-q8sh9
```

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

```
sh-5.1# chroot /host
```

- c. Display the UID being used by the node:

```
sh-5.1# lsns -t user
```

Example command

```

      NS TYPE NPROCS  PID USER    COMMAND
4026531837 user   233   1 root    /usr/lib/systemd/systemd --switched-root --system -
-deserialize 28
4026532447 user     1 4767 2908816384 /usr/bin/coreutils --coreutils-prog-
shebang=sleep /usr/bin/sleep 1000 ❶

```

- ❶ The UID should be different from what you set in the pod specification.

- d. Exit the debug session by using the following commands:

```
sh-5.1# exit
```

```
sh-5.1# exit
```

3. Check that the **/proc** file system is mounted into container as **unmasked**, as indicated by read/write permission (**rw**) in the output of the following command:

```
$ oc exec <pod_name> -- mount | grep /proc
```

Example output

```
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
```

Additional resources

- [Managing security context constraints](#)
- [OpenShift CLI administrator command reference](#)

CHAPTER 3. AUTOMATICALLY SCALING PODS WITH THE CUSTOM METRICS AUTOSCALER OPERATOR

3.1. RELEASE NOTES

3.1.1. Custom Metrics Autoscaler Operator release notes

The release notes for the Custom Metrics Autoscaler Operator for Red Hat OpenShift describe new features and enhancements, deprecated features, and known issues.

The Custom Metrics Autoscaler Operator uses the Kubernetes-based Event Driven Autoscaler (KEDA) and is built on top of the OpenShift Container Platform horizontal pod autoscaler (HPA).



NOTE

The Custom Metrics Autoscaler Operator for Red Hat OpenShift is provided as an installable component, with a distinct release cycle from the core OpenShift Container Platform. The [Red Hat OpenShift Container Platform Life Cycle Policy](#) outlines release compatibility.

3.1.1.1. Supported versions

The following table defines the Custom Metrics Autoscaler Operator versions for each OpenShift Container Platform version.

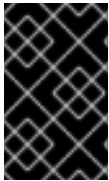
Version	OpenShift Container Platform version	General availability
2.17.2-2	4.20	General availability
2.17.2-2	4.19	General availability
2.17.2-2	4.18	General availability
2.17.2-2	4.17	General availability
2.17.2-2	4.16	General availability
2.17.2-2	4.15	General availability
2.17.2-2	4.14	General availability
2.17.2-2	4.13	General availability
2.17.2-2	4.12	General availability

3.1.1.2. Custom Metrics Autoscaler Operator 2.17.2-2 release notes

Issued: 21 October 2025

This release of the Custom Metrics Autoscaler Operator 2.17.2-2 is a rebuild of the 2.17.2 version of the Custom Metrics Autoscaler Operator using a newer base image and Go compiler. There are no code changes to the Custom Metrics Autoscaler Operator. The following advisory is available for the Custom Metrics Autoscaler Operator:

- [RHBA-2025:18914](#)



IMPORTANT

Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of Kubernetes-based Event Driven Autoscaler (KEDA).

3.1.2. Release notes for past releases of the Custom Metrics Autoscaler Operator

The following release notes are for previous versions of the Custom Metrics Autoscaler Operator.

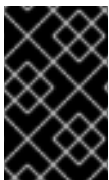
For the current version, see [Custom Metrics Autoscaler Operator release notes](#).

3.1.2.1. Custom Metrics Autoscaler Operator 2.17.2 release notes

Issued: 25 September 2025

This release of the Custom Metrics Autoscaler Operator 2.17.2 addresses Common Vulnerabilities and Exposures (CVEs). The following advisory is available for the Custom Metrics Autoscaler Operator:

- [RHSA-2025:16124](#)



IMPORTANT

Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of Kubernetes-based Event Driven Autoscaler (KEDA).

3.1.2.1.1. New features and enhancements

3.1.2.1.1.1. The KEDA controller is automatically created during installation

The KEDA controller is now automatically created when you install the Custom Metrics Autoscaler Operator. Previously, you needed to manually create the KEDA controller. You can edit the automatically-created KEDA controller, as needed.

3.1.2.1.1.2. Support for the Kubernetes workload trigger

The Cluster Metrics Autoscaler Operator now supports using the Kubernetes workload trigger to scale pods based on the number of pods matching a specific label selector.

3.1.2.1.1.3. Support for bound service account tokens

The Cluster Metrics Autoscaler Operator now supports bound service account tokens. Previously, the Operator supported only legacy service account tokens, which are being phased out in favor of bound service account tokens for security reasons.

3.1.2.1.2. Bug fixes

- Previously, the KEDA controller did not support volume mounts. As a result, you could not use Kerberos with the Kafka scaler. With this fix, the KEDA controller now supports volume mounts. ([OCPBUGS-42559](#))
- Previously, the KEDA version in the **keda-operator** deployment object log reported that the Custom Metrics Autoscaler Operator was based on an incorrect KEDA version. With this fix, the correct KEDA version is reported in the log. ([OCPBUGS-58129](#))

Additional resources

- [Editing the Keda Controller CR](#)
- [Understanding the Kubernetes workload trigger](#)
- [Understanding custom metrics autoscaler trigger authentications](#)

3.1.2.2. Custom Metrics Autoscaler Operator 2.15.1-4 release notes

Issued: 31 March 2025

This release of the Custom Metrics Autoscaler Operator 2.15.1-4 addresses Common Vulnerabilities and Exposures (CVEs). The following advisory is available for the Custom Metrics Autoscaler Operator:

- [RHSA-2025:3501](#)



IMPORTANT

Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of Kubernetes-based Event Driven Autoscaler (KEDA).

3.1.2.2.1. New features and enhancements

3.1.2.2.1.1. CMA multi-arch builds

With this version of the Custom Metrics Autoscaler Operator, you can now install and run the Operator on an ARM64 OpenShift Container Platform cluster.

3.1.2.3. Custom Metrics Autoscaler Operator 2.14.1-467 release notes

This release of the Custom Metrics Autoscaler Operator 2.14.1-467 provides a CVE and a bug fix for running the Operator in an OpenShift Container Platform cluster. The following advisory is available for the [RHSA-2024:7348](#).



IMPORTANT

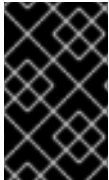
Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of Kubernetes-based Event Driven Autoscaler (KEDA).

3.1.2.3.1. Bug fixes

- Previously, the root file system of the Custom Metrics Autoscaler Operator pod was writable, which is unnecessary and could present security issues. This update makes the pod root file system read-only, which addresses the potential security issue. ([OCPBUGS-37989](#))

3.1.2.4. Custom Metrics Autoscaler Operator 2.14.1-454 release notes

This release of the Custom Metrics Autoscaler Operator 2.14.1-454 provides a CVE, a new feature, and bug fixes for running the Operator in an OpenShift Container Platform cluster. The following advisory is available for the [RHBA-2024:5865](#).



IMPORTANT

Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of Kubernetes-based Event Driven Autoscaler (KEDA).

3.1.2.4.1. New features and enhancements

3.1.2.4.1.1. Support for the Cron trigger with the Custom Metrics Autoscaler Operator

The Custom Metrics Autoscaler Operator can now use the Cron trigger to scale pods based on an hourly schedule. When your specified time frame starts, the Custom Metrics Autoscaler Operator scales pods to your desired amount. When the time frame ends, the Operator scales back down to the previous level.

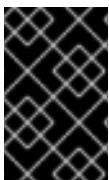
For more information, see [Understanding the Cron trigger](#).

3.1.2.4.2. Bug fixes

- Previously, if you made changes to audit configuration parameters in the **KedaController** custom resource, the **keda-metrics-server-audit-policy** config map would not get updated. As a consequence, you could not change the audit configuration parameters after the initial deployment of the Custom Metrics Autoscaler. With this fix, changes to the audit configuration now render properly in the config map, allowing you to change the audit configuration any time after installation. ([OCPBUGS-32521](#))

3.1.2.5. Custom Metrics Autoscaler Operator 2.13.1 release notes

This release of the Custom Metrics Autoscaler Operator 2.13.1-421 provides a new feature and a bug fix for running the Operator in an OpenShift Container Platform cluster. The following advisory is available for the [RHBA-2024:4837](#).



IMPORTANT

Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of Kubernetes-based Event Driven Autoscaler (KEDA).

3.1.2.5.1. New features and enhancements

3.1.2.5.1.1. Support for custom certificates with the Custom Metrics Autoscaler Operator

The Custom Metrics Autoscaler Operator can now use custom service CA certificates to connect

securely to TLS-enabled metrics sources, such as an external Kafka cluster or an external Prometheus service. By default, the Operator uses automatically-generated service certificates to connect to on-cluster services only. There is a new field in the **KedaController** object that allows you to load custom server CA certificates for connecting to external services by using config maps.

For more information, see [Custom CA certificates for the Custom Metrics Autoscaler](#).

3.1.2.5.2. Bug fixes

- Previously, the **custom-metrics-autoscaler** and **custom-metrics-autoscaler-adapter** images were missing time zone information. As a consequence, scaled objects with **cron** triggers failed to work because the controllers were unable to find time zone information. With this fix, the image builds are updated to include time zone information. As a result, scaled objects containing **cron** triggers now function properly. Scaled objects containing **cron** triggers are currently not supported for the custom metrics autoscaler. ([OCPBUGS-34018](#))

3.1.2.6. Custom Metrics Autoscaler Operator 2.12.1-394 release notes

This release of the Custom Metrics Autoscaler Operator 2.12.1-394 provides a bug fix for running the Operator in an OpenShift Container Platform cluster. The following advisory is available for the [RHSA-2024:2901](#).



IMPORTANT

Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of Kubernetes-based Event Driven Autoscaler (KEDA).

3.1.2.6.1. Bug fixes

- Previously, the **protojson.Unmarshal** function entered into an infinite loop when unmarshaling certain forms of invalid JSON. This condition could occur when unmarshaling into a message that contains a **google.protobuf.Any** value or when the **UnmarshalOptions.DiscardUnknown** option is set. This release fixes this issue. ([OCPBUGS-30305](#))
- Previously, when parsing a multipart form, either explicitly with the **Request.ParseMultipartForm** method or implicitly with the **Request.FormValue**, **Request.PostFormValue**, or **Request.FormFile** method, the limits on the total size of the parsed form were not applied to the memory consumed. This could cause memory exhaustion. With this fix, the parsing process now correctly limits the maximum size of form lines while reading a single form line. ([OCPBUGS-30360](#))
- Previously, when following an HTTP redirect to a domain that is not on a matching subdomain or on an exact match of the initial domain, an HTTP client would not forward sensitive headers, such as **Authorization** or **Cookie**. For example, a redirect from **example.com** to **www.example.com** would forward the **Authorization** header, but a redirect to **www.example.org** would not forward the header. This release fixes this issue. ([OCPBUGS-30365](#))
- Previously, verifying a certificate chain that contains a certificate with an unknown public key algorithm caused the certificate verification process to panic. This condition affected all crypto and Transport Layer Security (TLS) clients and servers that set the **Config.ClientAuth**

parameter to the **VerifyClientCertIfGiven** or **RequireAndVerifyClientCert** value. The default behavior is for TLS servers to not verify client certificates. This release fixes this issue. ([OCPBUGS-30370](#))

- Previously, if errors returned from the **MarshalJSON** method contained user-controlled data, an attacker could have used the data to break the contextual auto-escaping behavior of the HTML template package. This condition would allow for subsequent actions to inject unexpected content into the templates. This release fixes this issue. ([OCPBUGS-30397](#))
- Previously, the **net/http** and **golang.org/x/net/http2** Go packages did not limit the number of **CONTINUATION** frames for an HTTP/2 request. This condition could result in excessive CPU consumption. This release fixes this issue. ([OCPBUGS-30894](#))

3.1.2.7. Custom Metrics Autoscaler Operator 2.12.1-384 release notes

This release of the Custom Metrics Autoscaler Operator 2.12.1-384 provides a bug fix for running the Operator in an OpenShift Container Platform cluster. The following advisory is available for the [RHBA-2024:2043](#).



IMPORTANT

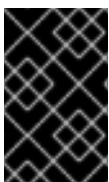
Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of KEDA.

3.1.2.7.1. Bug fixes

- Previously, the **custom-metrics-autoscaler** and **custom-metrics-autoscaler-adapter** images were missing time zone information. As a consequence, scaled objects with **cron** triggers failed to work because the controllers were unable to find time zone information. With this fix, the image builds are updated to include time zone information. As a result, scaled objects containing **cron** triggers now function properly. ([OCPBUGS-32395](#))

3.1.2.8. Custom Metrics Autoscaler Operator 2.12.1-376 release notes

This release of the Custom Metrics Autoscaler Operator 2.12.1-376 provides security updates and bug fixes for running the Operator in an OpenShift Container Platform cluster. The following advisory is available for the [RHSA-2024:1812](#).



IMPORTANT

Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of KEDA.

3.1.2.8.1. Bug fixes

- Previously, if invalid values such as nonexistent namespaces were specified in scaled object metadata, the underlying scaler clients would not free, or close, their client descriptors, resulting in a slow memory leak. This fix properly closes the underlying client descriptors when there are errors, preventing memory from leaking. ([OCPBUGS-30145](#))
- Previously the **ServiceMonitor** custom resource (CR) for the **keda-metrics-apiserver** pod was not functioning, because the CR referenced an incorrect metrics port name of **http**. This fix

corrects the **ServiceMonitor** CR to reference the proper port name of **metrics**. As a result, the Service Monitor functions properly. ([OCPBUGS-25806](#))

3.1.2.9. Custom Metrics Autoscaler Operator 2.11.2-322 release notes

This release of the Custom Metrics Autoscaler Operator 2.11.2-322 provides security updates and bug fixes for running the Operator in an OpenShift Container Platform cluster. The following advisory is available for the [RHSA-2023:6144](#).



IMPORTANT

Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of KEDA.

3.1.2.9.1. Bug fixes

- Because the Custom Metrics Autoscaler Operator version 3.11.2-311 was released without a required volume mount in the Operator deployment, the Custom Metrics Autoscaler Operator pod would restart every 15 minutes. This fix adds the required volume mount to the Operator deployment. As a result, the Operator no longer restarts every 15 minutes. ([OCPBUGS-22361](#))

3.1.2.10. Custom Metrics Autoscaler Operator 2.11.2-311 release notes

This release of the Custom Metrics Autoscaler Operator 2.11.2-311 provides new features and bug fixes for running the Operator in an OpenShift Container Platform cluster. The components of the Custom Metrics Autoscaler Operator 2.11.2-311 were released in [RHBA-2023:5981](#).



IMPORTANT

Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of KEDA.

3.1.2.10.1. New features and enhancements

3.1.2.10.1.1. Red Hat OpenShift Service on AWS and OpenShift Dedicated are now supported

The Custom Metrics Autoscaler Operator 2.11.2-311 can be installed on Red Hat OpenShift Service on AWS and OpenShift Dedicated managed clusters. Previous versions of the Custom Metrics Autoscaler Operator could be installed only in the **openshift-keda** namespace. This prevented the Operator from being installed on Red Hat OpenShift Service on AWS and OpenShift Dedicated clusters. This version of Custom Metrics Autoscaler allows installation to other namespaces such as **openshift-operators** or **keda**, enabling installation into Red Hat OpenShift Service on AWS and OpenShift Dedicated clusters.

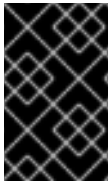
3.1.2.10.2. Bug fixes

- Previously, if the Custom Metrics Autoscaler Operator was installed and configured, but not in use, the OpenShift CLI reported the **couldn't get resource list for external.metrics.k8s.io/v1beta1: Got empty response for: external.metrics.k8s.io/v1beta1** error after any **oc** command was entered. The message, although harmless, could have caused confusion. With this fix, the **Got empty response for: external.metrics...** error no longer appears inappropriately. ([OCPBUGS-15779](#))

- Previously, any annotation or label change to objects managed by the Custom Metrics Autoscaler were reverted by Custom Metrics Autoscaler Operator any time the Keda Controller was modified, for example after a configuration change. This caused continuous changing of labels in your objects. The Custom Metrics Autoscaler now uses its own annotation to manage labels and annotations, and annotation or label are no longer inappropriately reverted. ([OCPBUGS-15590](#))

3.1.2.11. Custom Metrics Autoscaler Operator 2.10.1-267 release notes

This release of the Custom Metrics Autoscaler Operator 2.10.1-267 provides new features and bug fixes for running the Operator in an OpenShift Container Platform cluster. The components of the Custom Metrics Autoscaler Operator 2.10.1-267 were released in [RHBA-2023:4089](#).



IMPORTANT

Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of KEDA.

3.1.2.11.1. Bug fixes

- Previously, the **custom-metrics-autoscaler** and **custom-metrics-autoscaler-adapter** images did not contain time zone information. Because of this, scaled objects with cron triggers failed to work because the controllers were unable to find time zone information. With this fix, the image builds now include time zone information. As a result, scaled objects containing cron triggers now function properly. ([OCPBUGS-15264](#))
- Previously, the Custom Metrics Autoscaler Operator would attempt to take ownership of all managed objects, including objects in other namespaces and cluster-scoped objects. Because of this, the Custom Metrics Autoscaler Operator was unable to create the role binding for reading the credentials necessary to be an API server. This caused errors in the **kube-system** namespace. With this fix, the Custom Metrics Autoscaler Operator skips adding the **ownerReference** field to any object in another namespace or any cluster-scoped object. As a result, the role binding is now created without any errors. ([OCPBUGS-15038](#))
- Previously, the Custom Metrics Autoscaler Operator added an **ownerReferences** field to the **openshift-keda** namespace. While this did not cause functionality problems, the presence of this field could have caused confusion for cluster administrators. With this fix, the Custom Metrics Autoscaler Operator does not add the **ownerReference** field to the **openshift-keda** namespace. As a result, the **openshift-keda** namespace no longer has a superfluous **ownerReference** field. ([OCPBUGS-15293](#))
- Previously, if you used a Prometheus trigger configured with authentication method other than pod identity, and the **podIdentity** parameter was set to **none**, the trigger would fail to scale. With this fix, the Custom Metrics Autoscaler for OpenShift now properly handles the **none** pod identity provider type. As a result, a Prometheus trigger configured with authentication method other than pod identity, and the **podIdentity** parameter sset to **none** now properly scales. ([OCPBUGS-15274](#))

3.1.2.12. Custom Metrics Autoscaler Operator 2.10.1 release notes

This release of the Custom Metrics Autoscaler Operator 2.10.1 provides new features and bug fixes for running the Operator in an OpenShift Container Platform cluster. The components of the Custom Metrics Autoscaler Operator 2.10.1 were released in [RHEA-2023:3199](#).

**IMPORTANT**

Before installing this version of the Custom Metrics Autoscaler Operator, remove any previously installed Technology Preview versions or the community-supported version of KEDA.

3.1.2.12.1. New features and enhancements**3.1.2.12.1.1. Custom Metrics Autoscaler Operator general availability**

The Custom Metrics Autoscaler Operator is now generally available as of Custom Metrics Autoscaler Operator version 2.10.1.

**IMPORTANT**

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

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

3.1.2.12.1.2. Performance metrics

You can now use the Prometheus Query Language (PromQL) to query metrics on the Custom Metrics Autoscaler Operator.

3.1.2.12.1.3. Pausing the custom metrics autoscaling for scaled objects

You can now pause the autoscaling of a scaled object, as needed, and resume autoscaling when ready.

3.1.2.12.1.4. Replica fall back for scaled objects

You can now specify the number of replicas to fall back to if a scaled object fails to get metrics from the source.

3.1.2.12.1.5. Customizable HPA naming for scaled objects

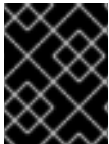
You can now specify a custom name for the horizontal pod autoscaler in scaled objects.

3.1.2.12.1.6. Activation and scaling thresholds

Because the horizontal pod autoscaler (HPA) cannot scale to or from 0 replicas, the Custom Metrics Autoscaler Operator does that scaling, after which the HPA performs the scaling. You can now specify when the HPA takes over autoscaling, based on the number of replicas. This allows for more flexibility with your scaling policies.

3.1.2.13. Custom Metrics Autoscaler Operator 2.8.2-174 release notes

This release of the Custom Metrics Autoscaler Operator 2.8.2-174 provides new features and bug fixes for running the Operator in an OpenShift Container Platform cluster. The components of the Custom Metrics Autoscaler Operator 2.8.2-174 were released in [RHEA-2023:1683](#).



IMPORTANT

The Custom Metrics Autoscaler Operator version 2.8.2-174 is a [Technology Preview](#) feature.

3.1.2.13.1. New features and enhancements

3.1.2.13.1.1. Operator upgrade support

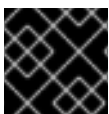
You can now upgrade from a prior version of the Custom Metrics Autoscaler Operator. See "Changing the update channel for an Operator" in the "Additional resources" for information on upgrading an Operator.

3.1.2.13.1.2. must-gather support

You can now collect data about the Custom Metrics Autoscaler Operator and its components by using the OpenShift Container Platform **must-gather** tool. Currently, the process for using the **must-gather** tool with the Custom Metrics Autoscaler is different than for other operators. See "Gathering debugging data in the "Additional resources" for more information.

3.1.2.14. Custom Metrics Autoscaler Operator 2.8.2 release notes

This release of the Custom Metrics Autoscaler Operator 2.8.2 provides new features and bug fixes for running the Operator in an OpenShift Container Platform cluster. The components of the Custom Metrics Autoscaler Operator 2.8.2 were released in [RHSA-2023:1042](#).



IMPORTANT

The Custom Metrics Autoscaler Operator version 2.8.2 is a [Technology Preview](#) feature.

3.1.2.14.1. New features and enhancements

3.1.2.14.1.1. Audit Logging

You can now gather and view audit logs for the Custom Metrics Autoscaler Operator and its associated components. Audit logs are security-relevant chronological sets of records that document the sequence of activities that have affected the system by individual users, administrators, or other components of the system.

3.1.2.14.1.2. Scale applications based on Apache Kafka metrics

You can now use the KEDA Apache kafka trigger/scaler to scale deployments based on an Apache Kafka topic.

3.1.2.14.1.3. Scale applications based on CPU metrics

You can now use the KEDA CPU trigger/scaler to scale deployments based on CPU metrics.

3.1.2.14.1.4. Scale applications based on memory metrics

You can now use the KEDA memory trigger/scaler to scale deployments based on memory metrics.

3.2. CUSTOM METRICS AUTOSCALER OPERATOR OVERVIEW

As a developer, you can use Custom Metrics Autoscaler Operator for Red Hat OpenShift to specify how OpenShift Container Platform should automatically increase or decrease the number of pods for a deployment, stateful set, custom resource, or job based on custom metrics that are not based only on CPU or memory.

The Custom Metrics Autoscaler Operator is an optional Operator, based on the Kubernetes Event Driven Autoscaler (KEDA), that allows workloads to be scaled using additional metrics sources other than pod metrics.

The custom metrics autoscaler currently supports only the Prometheus, CPU, memory, and Apache Kafka metrics.

The Custom Metrics Autoscaler Operator scales your pods up and down based on custom, external metrics from specific applications. Your other applications continue to use other scaling methods. You configure *triggers*, also known as *scalers*, which are the source of events and metrics that the custom metrics autoscaler uses to determine how to scale. The custom metrics autoscaler uses a metrics API to convert the external metrics to a form that OpenShift Container Platform can use. The custom metrics autoscaler creates a horizontal pod autoscaler (HPA) that performs the actual scaling.

To use the custom metrics autoscaler, you create a **ScaledObject** or **ScaledJob** object for a workload, which is a custom resource (CR) that defines the scaling metadata. You specify the deployment or job to scale, the source of the metrics to scale on (trigger), and other parameters such as the minimum and maximum replica counts allowed.



NOTE

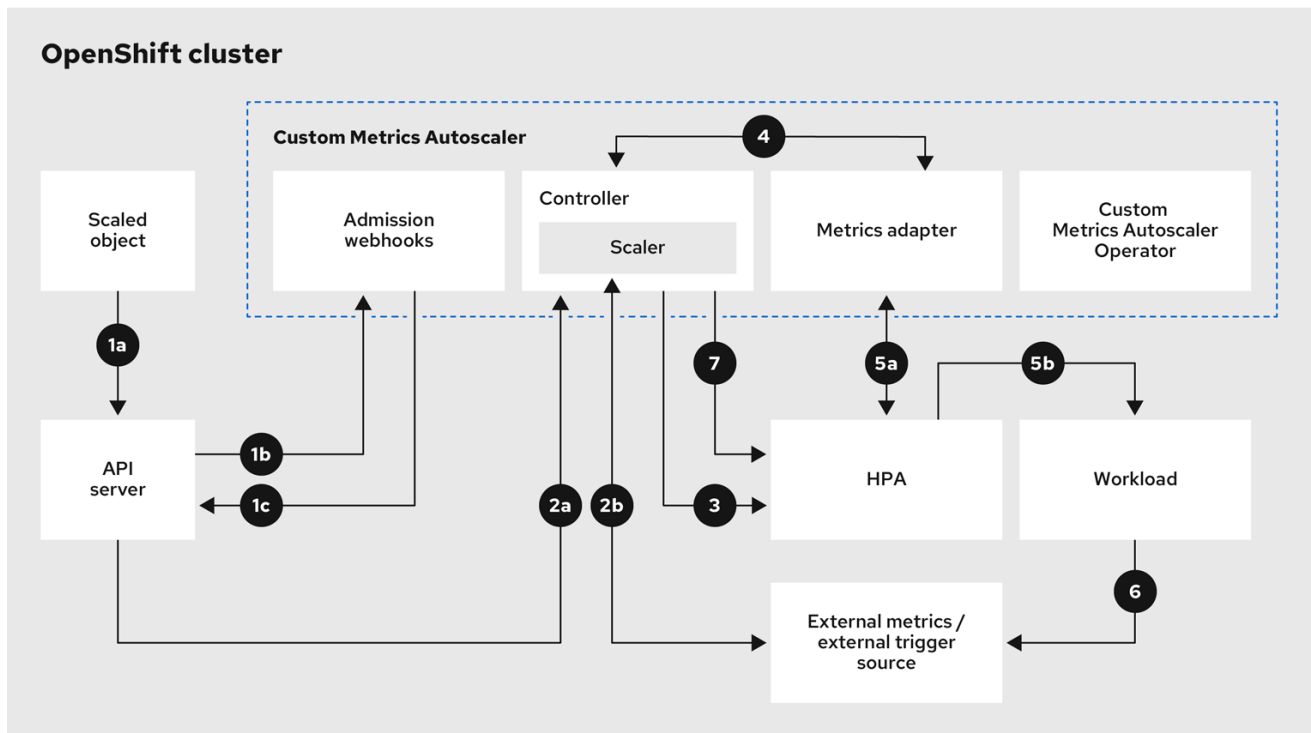
You can create only one scaled object or scaled job for each workload that you want to scale. Also, you cannot use a scaled object or scaled job and the horizontal pod autoscaler (HPA) on the same workload.

The custom metrics autoscaler, unlike the HPA, can scale to zero. If you set the **minReplicaCount** value in the custom metrics autoscaler CR to **0**, the custom metrics autoscaler scales the workload down from 1 to 0 replicas to or up from 0 replicas to 1. This is known as the *activation phase*. After scaling up to 1 replica, the HPA takes control of the scaling. This is known as the *scaling phase*.

Some triggers allow you to change the number of replicas that are scaled by the cluster metrics autoscaler. In all cases, the parameter to configure the activation phase always uses the same phrase, prefixed with *activation*. For example, if the **threshold** parameter configures scaling, **activationThreshold** would configure activation. Configuring the activation and scaling phases allows you more flexibility with your scaling policies. For example, you can configure a higher activation phase to prevent scaling up or down if the metric is particularly low.

The activation value has more priority than the scaling value in case of different decisions for each. For example, if the **threshold** is set to **10**, and the **activationThreshold** is **50**, if the metric reports **40**, the scaler is not active and the pods are scaled to zero even if the HPA requires 4 instances.

Figure 3.1. Custom metrics autoscaler workflow



565_OpenShift_0224

1. You create or modify a scaled object custom resource for a workload on a cluster. The object contains the scaling configuration for that workload. Prior to accepting the new object, the OpenShift API server sends it to the custom metrics autoscaler admission webhooks process to ensure that the object is valid. If validation succeeds, the API server persists the object.
2. The custom metrics autoscaler controller watches for new or modified scaled objects. When the OpenShift API server notifies the controller of a change, the controller monitors any external trigger sources, also known as data sources, that are specified in the object for changes to the metrics data. One or more scalers request scaling data from the external trigger source. For example, for a Kafka trigger type, the controller uses the Kafka scaler to communicate with a Kafka instance to obtain the data requested by the trigger.
3. The controller creates a horizontal pod autoscaler object for the scaled object. As a result, the Horizontal Pod Autoscaler (HPA) Operator starts monitoring the scaling data associated with the trigger. The HPA requests scaling data from the cluster OpenShift API server endpoint.
4. The OpenShift API server endpoint is served by the custom metrics autoscaler metrics adapter. When the metrics adapter receives a request for custom metrics, it uses a GRPC connection to the controller to request it for the most recent trigger data received from the scaler.
5. The HPA makes scaling decisions based upon the data received from the metrics adapter and scales the workload up or down by increasing or decreasing the replicas.
6. As it operates, a workload can affect the scaling metrics. For example, if a workload is scaled up to handle work in a Kafka queue, the queue size decreases after the workload processes all the work. As a result, the workload is scaled down.
7. If the metrics are in a range specified by the **minReplicaCount** value, the custom metrics autoscaler controller disables all scaling, and leaves the replica count at a fixed level. If the metrics exceed that range, the custom metrics autoscaler controller enables scaling and allows the HPA to scale the workload. While scaling is disabled, the HPA does not take any action.

3.2.1. Custom CA certificates for the Custom Metrics Autoscaler

By default, the Custom Metrics Autoscaler Operator uses automatically-generated service CA certificates to connect to on-cluster services.

If you want to use off-cluster services that require custom CA certificates, you can add the required certificates to a config map. Then, add the config map to the **KedaController** custom resource as described in [Installing the custom metrics autoscaler](#). The Operator loads those certificates on start-up and registers them as trusted by the Operator.

The config maps can contain one or more certificate files that contain one or more PEM-encoded CA certificates. Or, you can use separate config maps for each certificate file.



NOTE

If you later update the config map to add additional certificates, you must restart the **keda-operator-*** pod for the changes to take effect.

3.3. INSTALLING THE CUSTOM METRICS AUTOSCALER

You can use the OpenShift Container Platform web console to install the Custom Metrics Autoscaler Operator.

The installation creates the following five CRDs:

- **ClusterTriggerAuthentication**
- **KedaController**
- **ScaledJob**
- **ScaledObject**
- **TriggerAuthentication**

The installation process also creates the **KedaController** custom resource (CR). You can modify the default **KedaController** CR, if needed. For more information, see "Editing the Keda Controller CR".



NOTE

If you are installing a Custom Metrics Autoscaler Operator version lower than 2.17.2, you must manually create the Keda Controller CR. You can use the procedure described in "Editing the Keda Controller CR" to create the CR.

3.3.1. Installing the custom metrics autoscaler

You can use the following procedure to install the Custom Metrics Autoscaler Operator.

Prerequisites

- Remove any previously-installed Technology Preview versions of the Cluster Metrics Autoscaler Operator.
- Remove any versions of the community-based KEDA.

Also, remove the KEDA 1.x custom resource definitions by running the following commands:

```
$ oc delete crd scaledobjects.keda.k8s.io
```

```
$ oc delete crd triggerauthentications.keda.k8s.io
```

- Optional: If you need the Custom Metrics Autoscaler Operator to connect to off-cluster services, such as an external Kafka cluster or an external Prometheus service, put any required service CA certificates into a config map. The config map must exist in the same namespace where the Operator is installed. For example:

```
$ oc create configmap -n openshift-keda thanos-cert --from-file=ca-cert.pem
```

Procedure

1. In the OpenShift Container Platform web console, click **Ecosystem** → **Software Catalog**.
2. Choose **Custom Metrics Autoscaler** from the list of available Operators, and click **Install**.
3. On the **Install Operator** page, ensure that the **All namespaces on the cluster (default)** option is selected for **Installation Mode**. This installs the Operator in all namespaces.
4. Ensure that the **openshift-keda** namespace is selected for **Installed Namespace**. OpenShift Container Platform creates the namespace, if not present in your cluster.
5. Click **Install**.
6. Verify the installation by listing the Custom Metrics Autoscaler Operator components:
 - a. Navigate to **Workloads** → **Pods**.
 - b. Select the **openshift-keda** project from the drop-down menu and verify that the **custom-metrics-autoscaler-operator-*** pod is running.
 - c. Navigate to **Workloads** → **Deployments** to verify that the **custom-metrics-autoscaler-operator** deployment is running.
7. Optional: Verify the installation in the OpenShift CLI using the following commands:

```
$ oc get all -n openshift-keda
```

The output appears similar to the following:

Example output

```
NAME                                READY STATUS  RESTARTS  AGE
pod/custom-metrics-autoscaler-operator-5fd8d9ffd8-xt4xp  1/1    Running   0         18m
```

```
NAME                                READY UP-TO-DATE AVAILABLE AGE
deployment.apps/custom-metrics-autoscaler-operator  1/1    1          1         18m
```

```
NAME                                DESIRED CURRENT READY AGE
replicaset.apps/custom-metrics-autoscaler-operator-5fd8d9ffd8  1        1        1         18m
```

3.3.2. Editing the Keda Controller CR

You can use the following procedure to modify the **KedaController** custom resource (CR), which is automatically installed during the installation of the Custom Metrics Autoscaler Operator.

Procedure

1. In the OpenShift Container Platform web console, click **Ecosystem → Installed Operators**.
2. Click **Custom Metrics Autoscaler**.
3. On the **Operator Details** page, click the **KedaController** tab.
4. On the **KedaController** tab, click **Create KedaController** and edit the file.

```
kind: KedaController
apiVersion: keda.sh/v1alpha1
metadata:
  name: keda
  namespace: openshift-keda
spec:
  watchNamespace: " 1
  operator:
    logLevel: info 2
    logEncoder: console 3
    caConfigMaps: 4
    - thanos-cert
    - kafka-cert
    volumeMounts: 5
    - mountPath: /<path_to_directory>
      name: <name>
    volumes: 6
    - name: <volume_name>
      emptyDir:
        medium: Memory
  metricsServer:
    logLevel: '0' 7
    auditConfig: 8
    logFormat: "json"
    logOutputVolumeClaim: "persistentVolumeClaimName"
    policy:
      rules:
        - level: Metadata
      omitStages: ["RequestReceived"]
      omitManagedFields: false
    lifetime:
      maxAge: "2"
      maxBackup: "1"
      maxSize: "50"
  serviceAccount: {}
```

- 1 Specifies a single namespace in which the Custom Metrics Autoscaler Operator scales applications. Leave it blank or leave it empty to scale applications in all namespaces. This field should have a namespace or be empty. The default value is empty.

- 2 Specifies the level of verbosity for the Custom Metrics Autoscaler Operator log messages. The allowed values are **debug**, **info**, **error**. The default is **info**.
- 3 Specifies the logging format for the Custom Metrics Autoscaler Operator log messages. The allowed values are **console** or **json**. The default is **console**.
- 4 Optional: Specifies one or more config maps with CA certificates, which the Custom Metrics Autoscaler Operator can use to connect securely to TLS-enabled metrics sources.
- 5 Optional: Add the container mount path.
- 6 Optional: Add a **volumes** block to list each projected volume source.
- 7 Specifies the logging level for the Custom Metrics Autoscaler Metrics Server. The allowed values are **0** for **info** and **4** for **debug**. The default is **0**.
- 8 Activates audit logging for the Custom Metrics Autoscaler Operator and specifies the audit policy to use, as described in the "Configuring audit logging" section.

5. Click **Save** to save the changes.

3.4. UNDERSTANDING CUSTOM METRICS AUTOSCALER TRIGGERS

Triggers, also known as scalers, provide the metrics that the Custom Metrics Autoscaler Operator uses to scale your pods.

The custom metrics autoscaler currently supports the Prometheus, CPU, memory, Apache Kafka, and cron triggers.

You use a **ScaledObject** or **ScaledJob** custom resource to configure triggers for specific objects, as described in the sections that follow.

You can configure a certificate authority [to use with your scaled objects](#) or [for all scalers in the cluster](#).

3.4.1. Understanding the Prometheus trigger

You can scale pods based on Prometheus metrics, which can use the installed OpenShift Container Platform monitoring or an external Prometheus server as the metrics source. See "Configuring the custom metrics autoscaler to use OpenShift Container Platform monitoring" for information on the configurations required to use the OpenShift Container Platform monitoring as a source for metrics.



NOTE

If Prometheus is collecting metrics from the application that the custom metrics autoscaler is scaling, do not set the minimum replicas to **0** in the custom resource. If there are no application pods, the custom metrics autoscaler does not have any metrics to scale on.

Example scaled object with a Prometheus target

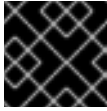
```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: prom-scaledobject
```

```

namespace: my-namespace
spec:
# ...
triggers:
- type: prometheus ❶
  metadata:
    serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092 ❷
    namespace: kedatest ❸
    metricName: http_requests_total ❹
    threshold: '5' ❺
    query: sum(rate(http_requests_total{job="test-app"}[1m])) ❻
    authModes: basic ❼
    cortexOrgID: my-org ❽
    ignoreNullValues: "false" ❾
    unsafeSsl: "false" ❿
    timeout: 1000 ⓫

```

- ❶ Specifies Prometheus as the trigger type.
- ❷ Specifies the address of the Prometheus server. This example uses OpenShift Container Platform monitoring.
- ❸ Optional: Specifies the namespace of the object you want to scale. This parameter is mandatory if using OpenShift Container Platform monitoring as a source for the metrics.
- ❹ Specifies the name to identify the metric in the **external.metrics.k8s.io** API. If you are using more than one trigger, all metric names must be unique.
- ❺ Specifies the value that triggers scaling. Must be specified as a quoted string value.
- ❻ Specifies the Prometheus query to use.
- ❼ Specifies the authentication method to use. Prometheus scalers support bearer authentication (**bearer**), basic authentication (**basic**), or TLS authentication (**tls**). You configure the specific authentication parameters in a trigger authentication, as discussed in a following section. As needed, you can also use a secret.
- ❽ Optional: Passes the **X-Scope-OrgID** header to multi-tenant [Cortex](#) or [Mimir](#) storage for Prometheus. This parameter is required only with multi-tenant Prometheus storage, to indicate which data Prometheus should return.
- ❾ Optional: Specifies how the trigger should proceed if the Prometheus target is lost.
 - If **true**, the trigger continues to operate if the Prometheus target is lost. This is the default behavior.
 - If **false**, the trigger returns an error if the Prometheus target is lost.
- ❿ Optional: Specifies whether the certificate check should be skipped. For example, you might skip the check if you are running in a test environment and using self-signed certificates at the Prometheus endpoint.
 - If **false**, the certificate check is performed. This is the default behavior.
 - If **true**, the certificate check is not performed.

**IMPORTANT**

Skipping the check is not recommended.

- 11** Optional: Specifies an HTTP request timeout in milliseconds for the HTTP client used by this Prometheus trigger. This value overrides any global timeout setting.

3.4.1.1. Configuring GPU-based autoscaling with Prometheus and DCGM metrics

You can use the Custom Metrics Autoscaler with NVIDIA Data Center GPU Manager (DCGM) metrics to scale workloads based on GPU utilization. This is particularly useful for AI and machine learning workloads that require GPU resources.

Example scaled object with a Prometheus target for GPU-based autoscaling

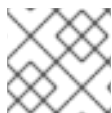
```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: gpu-scaledobject
  namespace: my-namespace
spec:
  scaleTargetRef:
    kind: Deployment
    name: gpu-deployment
  minReplicaCount: 1 1
  maxReplicaCount: 5 2
  triggers:
    - type: prometheus
      metadata:
        serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
        namespace: my-namespace
        metricName: gpu_utilization
        threshold: '90' 3
        query: SUM(DCGM_FI_DEV_GPU_UTIL{instance=~".+", gpu=~".+"}) 4
        authModes: bearer
      authenticationRef:
        name: keda-trigger-auth-prometheus
```

- 1** Specifies the minimum number of replicas to maintain. For GPU workloads, this should not be set to **0** to ensure that metrics continue to be collected.
- 2** Specifies the maximum number of replicas allowed during scale-up operations.
- 3** Specifies the GPU utilization percentage threshold that triggers scaling. When the average GPU utilization exceeds 90%, the autoscaler scales up the deployment.
- 4** Specifies a Prometheus query using NVIDIA DCGM metrics to monitor GPU utilization across all GPU devices. The **DCGM_FI_DEV_GPU_UTIL** metric provides GPU utilization percentages.

3.4.1.2. Configuring the custom metrics autoscaler to use OpenShift Container Platform monitoring

You can use the installed OpenShift Container Platform Prometheus monitoring as a source for the metrics used by the custom metrics autoscaler. However, there are some additional configurations you must perform.

For your scaled objects to be able to read the OpenShift Container Platform Prometheus metrics, you must use a trigger authentication or a cluster trigger authentication in order to provide the authentication information required. The following procedure differs depending on which trigger authentication method you use. For more information on trigger authentications, see "Understanding custom metrics autoscaler trigger authentications".



NOTE

These steps are not required for an external Prometheus source.

You must perform the following tasks, as described in this section:

- Create a service account.
- Create the trigger authentication.
- Create a role.
- Add that role to the service account.
- Reference the token in the trigger authentication object used by Prometheus.

Prerequisites

- OpenShift Container Platform monitoring must be installed.
- Monitoring of user-defined workloads must be enabled in OpenShift Container Platform monitoring, as described in the **Creating a user-defined workload monitoring config map** section.
- The Custom Metrics Autoscaler Operator must be installed.

Procedure

1. Change to the appropriate project:

```
$ oc project <project_name> 1
```

- 1 Specifies one of the following projects:

- If you are using a trigger authentication, specify the project with the object you want to scale.
- If you are using a cluster trigger authentication, specify the **openshift-keda** project.

2. Create a service account if your cluster does not have one:

- a. Create a **service account** object by using the following command:

```
$ oc create serviceaccount thanos 1
```

- 1 Specifies the name of the service account.

3. Create a trigger authentication with the service account token:

- a. Create a YAML file similar to the following:

```
apiVersion: keda.sh/v1alpha1
kind: <authentication_method> 1
metadata:
  name: keda-trigger-auth-prometheus
spec:
  boundServiceAccountToken: 2
  - parameter: bearerToken 3
    serviceAccountName: thanos 4
```

- 1 Specifies one of the following trigger authentication methods:
 - If you are using a trigger authentication, specify **TriggerAuthentication**. This example configures a trigger authentication.
 - If you are using a cluster trigger authentication, specify **ClusterTriggerAuthentication**.
- 2 Specifies that this trigger authentication uses a bound service account token for authorization when connecting to the metrics endpoint.
- 3 Specifies the authentication parameter to supply by using the token. Here, the example uses bearer authentication.
- 4 Specifies the name of the service account to use.

- b. Create the CR object:

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

4. Create a role for reading Thanos metrics:

- a. Create a YAML file with the following parameters:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: thanos-metrics-reader
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
- apiGroups:
  - metrics.k8s.io
  resources:
  - pods
```

```
- nodes
verbs:
- get
- list
- watch
```

b. Create the CR object:

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

5. Create a role binding for reading Thanos metrics:

a. Create a YAML file similar to the following:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: <binding_type> 1
metadata:
  name: thanos-metrics-reader 2
  namespace: my-project 3
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: thanos-metrics-reader
subjects:
- kind: ServiceAccount
  name: thanos 4
  namespace: <namespace_name> 5
```

- 1** Specifies one of the following object types:
 - If you are using a trigger authentication, specify **RoleBinding**.
 - If you are using a cluster trigger authentication, specify **ClusterRoleBinding**.
- 2** Specifies the name of the role you created.
- 3** Specifies one of the following projects:
 - If you are using a trigger authentication, specify the project with the object you want to scale.
 - If you are using a cluster trigger authentication, specify the **openshift-keda** project.
- 4** Specifies the name of the service account to bind to the role.
- 5** Specifies the project where you previously created the service account.

b. Create the CR object:

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

You can now deploy a scaled object or scaled job to enable autoscaling for your application, as described in "Understanding how to add custom metrics autoscalers". To use OpenShift Container Platform monitoring as the source, in the trigger, or scaler, you must include the following parameters:

- **triggers.type** must be **prometheus**
- **triggers.metadata.serverAddress** must be **https://thanos-querier.openshift-monitoring.svc.cluster.local:9092**
- **triggers.metadata.authModes** must be **bearer**
- **triggers.metadata.namespace** must be set to the namespace of the object to scale
- **triggers.authenticationRef** must point to the trigger authentication resource specified in the previous step

Additional resources

- [Understanding custom metrics autoscaler trigger authentications](#)

3.4.2. Understanding the CPU trigger

You can scale pods based on CPU metrics. This trigger uses cluster metrics as the source for metrics.

The custom metrics autoscaler scales the pods associated with an object to maintain the CPU usage that you specify. The autoscaler increases or decreases the number of replicas between the minimum and maximum numbers to maintain the specified CPU utilization across all pods. The memory trigger considers the memory utilization of the entire pod. If the pod has multiple containers, the memory trigger considers the total memory utilization of all containers in the pod.



NOTE

- This trigger cannot be used with the **ScaledJob** custom resource.
- When using a memory trigger to scale an object, the object does not scale to **0**, even if you are using multiple triggers.

Example scaled object with a CPU target

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: cpu-scaledobject
  namespace: my-namespace
spec:
  # ...
  triggers:
  - type: cpu 1
    metricType: Utilization 2
    metadata:
      value: '60' 3
  minReplicaCount: 1 4
```

- 1 Specifies CPU as the trigger type.

- 2 Specifies the type of metric to use, either **Utilization** or **AverageValue**.
- 3 Specifies the value that triggers scaling. Must be specified as a quoted string value.
 - When using **Utilization**, the target value is the average of the resource metrics across all relevant pods, represented as a percentage of the requested value of the resource for the pods.
 - When using **AverageValue**, the target value is the average of the metrics across all relevant pods.
- 4 Specifies the minimum number of replicas when scaling down. For a CPU trigger, enter a value of **1** or greater, because the HPA cannot scale to zero if you are using only CPU metrics.

3.4.3. Understanding the memory trigger

You can scale pods based on memory metrics. This trigger uses cluster metrics as the source for metrics.

The custom metrics autoscaler scales the pods associated with an object to maintain the average memory usage that you specify. The autoscaler increases and decreases the number of replicas between the minimum and maximum numbers to maintain the specified memory utilization across all pods. The memory trigger considers the memory utilization of entire pod. If the pod has multiple containers, the memory utilization is the sum of all of the containers.



NOTE

- This trigger cannot be used with the **ScaledJob** custom resource.
- When using a memory trigger to scale an object, the object does not scale to **0**, even if you are using multiple triggers.

Example scaled object with a memory target

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: memory-scaledobject
  namespace: my-namespace
spec:
  # ...
  triggers:
  - type: memory 1
    metricType: Utilization 2
    metadata:
      value: '60' 3
      containerName: api 4
```

- 1 Specifies memory as the trigger type.
- 2 Specifies the type of metric to use, either **Utilization** or **AverageValue**.
- 3 Specifies the value that triggers scaling. Must be specified as a quoted string value.

- When using **Utilization**, the target value is the average of the resource metrics across all relevant pods, represented as a percentage of the requested value of the resource for the pods.
- When using **AverageValue**, the target value is the average of the metrics across all relevant pods.

- 4 Optional: Specifies an individual container to scale, based on the memory utilization of only that container, rather than the entire pod. In this example, only the container named **api** is to be scaled.

3.4.4. Understanding the Kafka trigger

You can scale pods based on an Apache Kafka topic or other services that support the Kafka protocol. The custom metrics autoscaler does not scale higher than the number of Kafka partitions, unless you set the **allowIdleConsumers** parameter to **true** in the scaled object or scaled job.



NOTE

If the number of consumer groups exceeds the number of partitions in a topic, the extra consumer groups remain idle. To avoid this, by default the number of replicas does not exceed:

- The number of partitions on a topic, if a topic is specified
- The number of partitions of all topics in the consumer group, if no topic is specified
- The **maxReplicaCount** specified in scaled object or scaled job CR

You can use the **allowIdleConsumers** parameter to disable these default behaviors.

Example scaled object with a Kafka target

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: kafka-scaledobject
  namespace: my-namespace
spec:
  # ...
  triggers:
    - type: kafka 1
      metadata:
        topic: my-topic 2
        bootstrapServers: my-cluster-kafka-bootstrap.openshift-operators.svc:9092 3
        consumerGroup: my-group 4
        lagThreshold: '10' 5
        activationLagThreshold: '5' 6
        offsetResetPolicy: latest 7
        allowIdleConsumers: true 8
        scaleToZeroOnInvalidOffset: false 9
        excludePersistentLag: false 10
```

```

version: '1.0.0' 11
partitionLimitation: '1,2,10-20,31' 12
tls: enable 13

```

- ¹ Specifies Kafka as the trigger type.
- ² Specifies the name of the Kafka topic on which Kafka is processing the offset lag.
- ³ Specifies a comma-separated list of Kafka brokers to connect to.
- ⁴ Specifies the name of the Kafka consumer group used for checking the offset on the topic and processing the related lag.
- ⁵ Optional: Specifies the average target value that triggers scaling. Must be specified as a quoted string value. The default is **5**.
- ⁶ Optional: Specifies the target value for the activation phase. Must be specified as a quoted string value.
- ⁷ Optional: Specifies the Kafka offset reset policy for the Kafka consumer. The available values are: **latest** and **earliest**. The default is **latest**.
- ⁸ Optional: Specifies whether the number of Kafka replicas can exceed the number of partitions on a topic.
 - If **true**, the number of Kafka replicas can exceed the number of partitions on a topic. This allows for idle Kafka consumers.
 - If **false**, the number of Kafka replicas cannot exceed the number of partitions on a topic. This is the default.
- ⁹ Specifies how the trigger behaves when a Kafka partition does not have a valid offset.
 - If **true**, the consumers are scaled to zero for that partition.
 - If **false**, the scaler keeps a single consumer for that partition. This is the default.
- ¹⁰ Optional: Specifies whether the trigger includes or excludes partition lag for partitions whose current offset is the same as the current offset of the previous polling cycle.
 - If **true**, the scaler excludes partition lag in these partitions.
 - If **false**, the trigger includes all consumer lag in all partitions. This is the default.
- ¹¹ Optional: Specifies the version of your Kafka brokers. Must be specified as a quoted string value. The default is **1.0.0**.
- ¹² Optional: Specifies a comma-separated list of partition IDs to scope the scaling on. If set, only the listed IDs are considered when calculating lag. Must be specified as a quoted string value. The default is to consider all partitions.
- ¹³ Optional: Specifies whether to use TLS client authentication for Kafka. The default is **disable**. For information on configuring TLS, see "Understanding custom metrics autoscaler trigger authentications".

3.4.5. Understanding the Cron trigger

You can scale pods based on a time range.

When the time range starts, the custom metrics autoscaler scales the pods associated with an object from the configured minimum number of pods to the specified number of desired pods. At the end of the time range, the pods are scaled back to the configured minimum. The time period must be configured in [cron format](#).

The following example scales the pods associated with this scaled object from **0** to **100** from 6:00 AM to 6:30 PM India Standard Time.

Example scaled object with a Cron trigger

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: cron-scaledobject
  namespace: default
spec:
  scaleTargetRef:
    name: my-deployment
  minReplicaCount: 0 1
  maxReplicaCount: 100 2
  cooldownPeriod: 300
  triggers:
    - type: cron 3
      metadata:
        timezone: Asia/Kolkata 4
        start: "0 6 * * *" 5
        end: "30 18 * * *" 6
        desiredReplicas: "100" 7
```

- ¹ Specifies the minimum number of pods to scale down to at the end of the time frame.
- ² Specifies the maximum number of replicas when scaling up. This value should be the same as **desiredReplicas**. The default is **100**.
- ³ Specifies a Cron trigger.
- ⁴ Specifies the timezone for the time frame. This value must be from the [IANA Time Zone Database](#).
- ⁵ Specifies the start of the time frame.
- ⁶ Specifies the end of the time frame.
- ⁷ Specifies the number of pods to scale to between the start and end of the time frame. This value should be the same as **maxReplicaCount**.

3.4.6. Understanding the Kubernetes workload trigger

You can scale pods based on the number of pods matching a specific label selector.

The Custom Metrics Autoscaler Operator tracks the number of pods with a specific label that are in the

same namespace, then calculates a *relation* based on the number of labeled pods to the pods for the scaled object. Using this relation, the Custom Metrics Autoscaler Operator scales the object according to the scaling policy in the **ScaledObject** or **ScaledJob** specification.

The pod counts includes pods with a **Succeeded** or **Failed** phase.

For example, if you have a **frontend** deployment and a **backend** deployment. You can use a **kubernetes-workload** trigger to scale the **backend** deployment based on the number of **frontend** pods. If number of **frontend** pods goes up, the Operator would scale the **backend** pods to maintain the specified ratio. In this example, if there are 10 pods with the **app=frontend** pod selector, the Operator scales the backend pods to 5 in order to maintain the **0.5** ratio set in the scaled object.

Example scaled object with a Kubernetes workload trigger

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: workload-scaledobject
  namespace: my-namespace
spec:
  triggers:
    - type: kubernetes-workload 1
      metadata:
        podSelector: 'app=frontend' 2
        value: '0.5' 3
        activationValue: '3.1' 4
```

- 1** Specifies a Kubernetes workload trigger.
- 2** Specifies one or more pod selectors and/or set-based selectors, separated with commas, to use to get the pod count.
- 3** Specifies the target relation between the scaled workload and the number of pods that match the selector. The relation is calculated following the following formula:

$$\text{relation} = (\text{pods that match the selector}) / (\text{scaled workload pods})$$

- 4** Optional: Specifies the target value for scaler activation phase. The default is **0**.

3.5. UNDERSTANDING CUSTOM METRICS AUTOSCALER TRIGGER AUTHENTIFICATIONS

A trigger authentication allows you to include authentication information in a scaled object or a scaled job that can be used by the associated containers. You can use trigger authentications to pass OpenShift Container Platform secrets, platform-native pod authentication mechanisms, environment variables, and so on.

You define a **TriggerAuthentication** object in the same namespace as the object that you want to scale. That trigger authentication can be used only by objects in that namespace.

Alternatively, to share credentials between objects in multiple namespaces, you can create a **ClusterTriggerAuthentication** object that can be used across all namespaces.

Trigger authentications and cluster trigger authentication use the same configuration. However, a cluster trigger authentication requires an additional **kind** parameter in the authentication reference of the scaled object.

Example trigger authentication that uses a bound service account token

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: secret-triggerauthentication
  namespace: my-namespace ❶
spec:
  boundServiceAccountToken: ❷
  - parameter: bearerToken
    serviceAccountName: thanos ❸
```

- ❶ Specifies the namespace of the object you want to scale.
- ❷ Specifies that this trigger authentication uses a bound service account token for authorization when connecting to the metrics endpoint.
- ❸ Specifies the name of the service account to use.

Example cluster trigger authentication that uses a bound service account token

```
kind: ClusterTriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: bound-service-account-token-triggerauthentication ❶
spec:
  boundServiceAccountToken: ❷
  - parameter: bearerToken
    serviceAccountName: thanos ❸
```

- ❶ Specifies the namespace of the object you want to scale.
- ❷ Specifies that this cluster trigger authentication uses a bound service account token for authorization when connecting to the metrics endpoint.
- ❸ Specifies the name of the service account to use.

Example trigger authentication that uses a secret for Basic authentication

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: secret-triggerauthentication
  namespace: my-namespace ❶
spec:
  secretTargetRef: ❷
  - parameter: username ❸
    name: my-basic-secret ❹
```

```

key: username 5
- parameter: password
name: my-basic-secret
key: password

```

- 1 Specifies the namespace of the object you want to scale.
- 2 Specifies that this trigger authentication uses a secret for authorization when connecting to the metrics endpoint.
- 3 Specifies the authentication parameter to supply by using the secret.
- 4 Specifies the name of the secret to use. See the following example secret for Basic authentication.
- 5 Specifies the key in the secret to use with the specified parameter.

Example secret for Basic authentication

```

apiVersion: v1
kind: Secret
metadata:
  name: my-basic-secret
  namespace: default
data:
  username: "dXNlcm5hbWU=" 1
  password: "cGFzc3dvcmQ="

```

- 1 User name and password to supply to the trigger authentication. The values in the **data** stanza must be base-64 encoded.

Example trigger authentication that uses a secret for CA details

```

kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: secret-triggerauthentication
  namespace: my-namespace 1
spec:
  secretTargetRef: 2
  - parameter: key 3
    name: my-secret 4
    key: client-key.pem 5
  - parameter: ca 6
    name: my-secret 7
    key: ca-cert.pem 8

```

- 1 Specifies the namespace of the object you want to scale.
- 2 Specifies that this trigger authentication uses a secret for authorization when connecting to the metrics endpoint.
- 3 Specifies the type of authentication to use.

- 4 Specifies the name of the secret to use.
- 5 Specifies the key in the secret to use with the specified parameter.
- 6 Specifies the authentication parameter for a custom CA when connecting to the metrics endpoint.
- 7 Specifies the name of the secret to use. See the following example secret with certificate authority (CA) details.
- 8 Specifies the key in the secret to use with the specified parameter.

Example secret with certificate authority (CA) details

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
  namespace: my-namespace
data:
  ca-cert.pem: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0... 1
  client-cert.pem: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0... 2
  client-key.pem: LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0t...
```

- 1 Specifies the TLS CA Certificate for authentication of the metrics endpoint. The value must be base-64 encoded.
- 2 Specifies the TLS certificates and key for TLS client authentication. The values must be base-64 encoded.

Example trigger authentication that uses a bearer token

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: token-triggerauthentication
  namespace: my-namespace 1
spec:
  secretTargetRef: 2
  - parameter: bearerToken 3
    name: my-secret 4
    key: bearerToken 5
```

- 1 Specifies the namespace of the object you want to scale.
- 2 Specifies that this trigger authentication uses a secret for authorization when connecting to the metrics endpoint.
- 3 Specifies the type of authentication to use.
- 4 Specifies the name of the secret to use. See the following example secret for a bearer token.
- 5 Specifies the key in the token to use with the specified parameter.

Example secret for a bearer token

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
  namespace: my-namespace
data:
  bearerToken: "<bearer_token>" ❶
```

- ❶ Specifies a bearer token to use with bearer authentication. The value must be base-64 encoded.

Example trigger authentication that uses an environment variable

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: env-var-triggerauthentication
  namespace: my-namespace ❶
spec:
  env: ❷
  - parameter: access_key ❸
    name: ACCESS_KEY ❹
  containerName: my-container ❺
```

- ❶ Specifies the namespace of the object you want to scale.
- ❷ Specifies that this trigger authentication uses environment variables for authorization when connecting to the metrics endpoint.
- ❸ Specify the parameter to set with this variable.
- ❹ Specify the name of the environment variable.
- ❺ Optional: Specify a container that requires authentication. The container must be in the same resource as referenced by **scaleTargetRef** in the scaled object.

Example trigger authentication that uses pod authentication providers

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: pod-id-triggerauthentication
  namespace: my-namespace ❶
spec:
  podIdentity: ❷
  provider: aws-eks ❸
```

- ❶ Specifies the namespace of the object you want to scale.
- ❷ Specifies that this trigger authentication uses a platform-native pod authentication when connecting to the metrics endpoint.

- 3 Specifies a pod identity. Supported values are **none**, **azure**, **gcp**, **aws-eks**, or **aws-kiam**. The default is **none**.

Additional resources

- [Understanding and creating service accounts](#)
- [Providing sensitive data to pods](#).

3.5.1. Using trigger authentications

You use trigger authentications and cluster trigger authentications by using a custom resource to create the authentication, then add a reference to a scaled object or scaled job.

Prerequisites

- The Custom Metrics Autoscaler Operator must be installed.
- If you are using a bound service account token, the service account must exist.
- If you are using a bound service account token, a role-based access control (RBAC) object that enables the Custom Metrics Autoscaler Operator to request service account tokens from the service account must exist.

```

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: keda-operator-token-creator
  namespace: <namespace_name> 1
rules:
- apiGroups:
  - ""
  resources:
  - serviceaccounts/token
  verbs:
  - create
  resourceNames:
  - thanos 2
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: keda-operator-token-creator-binding
  namespace: <namespace_name> 3
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: keda-operator-token-creator
subjects:
- kind: ServiceAccount
  name: keda-operator
  namespace: openshift-keda

```

- 1 Specifies the namespace of the service account.

- 2 Specifies the name of the service account.
 - 3 Specifies the namespace of the service account.
- If you are using a secret, the **Secret** object must exist.

Procedure

1. Create the **TriggerAuthentication** or **ClusterTriggerAuthentication** object.
 - a. Create a YAML file that defines the object:

Example trigger authentication with a bound service account token

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: prom-triggerauthentication
  namespace: my-namespace 1
spec:
  boundServiceAccountToken: 2
  - parameter: token
    serviceAccountName: thanos 3
```

- 1 Specifies the namespace of the object you want to scale.
- 2 Specifies that this trigger authentication uses a bound service account token for authorization when connecting to the metrics endpoint.
- 3 Specifies the name of the service account to use.

- b. Create the **TriggerAuthentication** object:

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

2. Create or edit a **ScaledObject** YAML file that uses the trigger authentication:
 - a. Create a YAML file that defines the object by running the following command:

Example scaled object with a trigger authentication

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: scaledobject
  namespace: my-namespace
spec:
  scaleTargetRef:
    name: example-deployment
  maxReplicaCount: 100
  minReplicaCount: 0
  pollingInterval: 30
  triggers:
```

```

- type: prometheus
  metadata:
    serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
    namespace: kedatest # replace <NAMESPACE>
    metricName: http_requests_total
    threshold: '5'
    query: sum(rate(http_requests_total{job="test-app"}[1m]))
    authModes: "basic"
  authenticationRef:
    name: prom-triggerauthentication 1
    kind: TriggerAuthentication 2

```

- 1** Specify the name of your trigger authentication object.
- 2** Specify **TriggerAuthentication**. **TriggerAuthentication** is the default.

Example scaled object with a cluster trigger authentication

```

apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: scaledobject
  namespace: my-namespace
spec:
  scaleTargetRef:
    name: example-deployment
  maxReplicaCount: 100
  minReplicaCount: 0
  pollingInterval: 30
  triggers:
    - type: prometheus
      metadata:
        serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
        namespace: kedatest # replace <NAMESPACE>
        metricName: http_requests_total
        threshold: '5'
        query: sum(rate(http_requests_total{job="test-app"}[1m]))
        authModes: "basic"
      authenticationRef:
        name: prom-cluster-triggerauthentication 1
        kind: ClusterTriggerAuthentication 2

```

- 1** Specify the name of your trigger authentication object.
- 2** Specify **ClusterTriggerAuthentication**.

b. Create the scaled object by running the following command:

```
$ oc apply -f <filename>
```


3.6. UNDERSTANDING HOW TO ADD CUSTOM METRICS AUTOSCALERS

To add a custom metrics autoscaler, create a **ScaledObject** custom resource for a deployment, stateful set, or custom resource. Create a **ScaledJob** custom resource for a job.

You can create only one scaled object for each workload that you want to scale. Also, you cannot use a scaled object and the horizontal pod autoscaler (HPA) on the same workload.

3.6.1. Adding a custom metrics autoscaler to a workload

You can create a custom metrics autoscaler for a workload that is created by a **Deployment**, **StatefulSet**, or **custom resource** object.

Prerequisites

- The Custom Metrics Autoscaler Operator must be installed.
- If you use a custom metrics autoscaler for scaling based on CPU or memory:
 - Your cluster administrator must have properly configured cluster metrics. You can use the **oc describe PodMetrics <pod-name>** command to determine if metrics are configured. If metrics are configured, the output appears similar to the following, with CPU and Memory displayed under Usage.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

Example output

```
Name:      openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace: openshift-kube-scheduler
Labels:    <none>
Annotations: <none>
API Version: metrics.k8s.io/v1beta1
Containers:
  Name: wait-for-host-port
  Usage:
    Memory: 0
  Name: scheduler
  Usage:
    Cpu: 8m
    Memory: 45440Ki
Kind:      PodMetrics
Metadata:
  Creation Timestamp: 2019-05-23T18:47:56Z
  Self Link:          /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-
scheduler/pods/openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
  Timestamp:          2019-05-23T18:47:56Z
  Window:              1m0s
  Events:              <none>
```

- The pods associated with the object you want to scale must include specified memory and CPU limits. For example:

Example pod spec

```

apiVersion: v1
kind: Pod
# ...
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
  resources:
    limits:
      memory: "128Mi"
      cpu: "500m"
# ...

```

Procedure

1. Create a YAML file similar to the following. Only the name **<2>**, object name **<4>**, and object kind **<5>** are required:

Example scaled object

```

apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "0" 1
  name: scaledobject 2
  namespace: my-namespace
spec:
  scaleTargetRef:
    apiVersion: apps/v1 3
    name: example-deployment 4
    kind: Deployment 5
    envSourceContainerName: .spec.template.spec.containers[0] 6
  cooldownPeriod: 200 7
  maxReplicaCount: 100 8
  minReplicaCount: 0 9
  metricsServer: 10
  auditConfig:
    logFormat: "json"
    logOutputVolumeClaim: "persistentVolumeClaimName"
    policy:
      rules:
      - level: Metadata
        omitStages: "RequestReceived"
        omitManagedFields: false
    lifetime:
      maxAge: "2"
      maxBackup: "1"
      maxSize: "50"
  fallback: 11
  failureThreshold: 3

```

```

replicas: 6
behavior: static 12
pollingInterval: 30 13
advanced:
  restoreToOriginalReplicaCount: false 14
  horizontalPodAutoscalerConfig:
    name: keda-hpa-scale-down 15
    behavior: 16
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
        - type: Percent
          value: 100
          periodSeconds: 15
  triggers:
    - type: prometheus 17
      metadata:
        serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
        namespace: kedatest
        metricName: http_requests_total
        threshold: '5'
        query: sum(rate(http_requests_total{job="test-app"}[1m]))
        authModes: basic
      authenticationRef: 18
        name: prom-triggerauthentication
        kind: TriggerAuthentication

```

- 1 Optional: Specifies that the Custom Metrics Autoscaler Operator is to scale the replicas to the specified value and stop autoscaling, as described in the "Pausing the custom metrics autoscaler for a workload" section.
- 2 Specifies a name for this custom metrics autoscaler.
- 3 Optional: Specifies the API version of the target resource. The default is **apps/v1**.
- 4 Specifies the name of the object that you want to scale.
- 5 Specifies the **kind** as **Deployment**, **StatefulSet** or **CustomResource**.
- 6 Optional: Specifies the name of the container in the target resource, from which the custom metrics autoscaler gets environment variables holding secrets and so forth. The default is **.spec.template.spec.containers[0]**.
- 7 Optional. Specifies the period in seconds to wait after the last trigger is reported before scaling the deployment back to **0** if the **minReplicaCount** is set to **0**. The default is **300**.
- 8 Optional: Specifies the maximum number of replicas when scaling up. The default is **100**.
- 9 Optional: Specifies the minimum number of replicas when scaling down.
- 10 Optional: Specifies the parameters for audit logs. as described in the "Configuring audit logging" section.
- 11

Optional: Specifies the number of replicas to fall back to if a scaler fails to get metrics from the source for the number of times defined by the **failureThreshold** parameter. For more

- 12 Optional: Specifies the replica count to be used if a fallback occurs. Enter one of the following options or omit the parameter:
 - Enter **static** to use the number of replicas specified by the **fallback.replicas** parameter. This is the default.
 - Enter **currentReplicas** to maintain the current number of replicas.
 - Enter **currentReplicasIfHigher** to maintain the current number of replicas, if that number is higher than the **fallback.replicas** parameter. If the current number of replicas is lower than the **fallback.replicas** parameter, use the **fallback.replicas** value.
 - Enter **currentReplicasIfLower** to maintain the current number of replicas, if that number is lower than the **fallback.replicas** parameter. If the current number of replicas is higher than the **fallback.replicas** parameter, use the **fallback.replicas** value.
- 13 Optional: Specifies the interval in seconds to check each trigger on. The default is **30**.
- 14 Optional: Specifies whether to scale back the target resource to the original replica count after the scaled object is deleted. The default is **false**, which keeps the replica count as it is when the scaled object is deleted.
- 15 Optional: Specifies a name for the horizontal pod autoscaler. The default is **keda-hpa-{scaled-object-name}**.
- 16 Optional: Specifies a scaling policy to use to control the rate to scale pods up or down, as described in the "Scaling policies" section.
- 17 Specifies the trigger to use as the basis for scaling, as described in the "Understanding the custom metrics autoscaler triggers" section. This example uses OpenShift Container Platform monitoring.
- 18 Optional: Specifies a trigger authentication or a cluster trigger authentication. For more information, see *Understanding the custom metrics autoscaler trigger authentication* in the *Additional resources* section.
 - Enter **TriggerAuthentication** to use a trigger authentication. This is the default.
 - Enter **ClusterTriggerAuthentication** to use a cluster trigger authentication.

2. Create the custom metrics autoscaler by running the following command:

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

Verification

- View the command output to verify that the custom metrics autoscaler was created:

```
$ oc get scaledobject <scaled_object_name>
```

Example output

NAME	SCALETARGETKIND	SCALETARGETNAME	MIN	MAX	TRIGGERS
AUTHENTICATION	READY	ACTIVE	FALLBACK	AGE	
scaledobject	apps/v1.Deployment	example-deployment	0	50	prometheus prom-triggerauthentication
True	True	True	17s		

Note the following fields in the output:

- **TRIGGERS:** Indicates the trigger, or scaler, that is being used.
- **AUTHENTICATION:** Indicates the name of any trigger authentication being used.
- **READY:** Indicates whether the scaled object is ready to start scaling:
 - If **True**, the scaled object is ready.
 - If **False**, the scaled object is not ready because of a problem in one or more of the objects you created.
- **ACTIVE:** Indicates whether scaling is taking place:
 - If **True**, scaling is taking place.
 - If **False**, scaling is not taking place because there are no metrics or there is a problem in one or more of the objects you created.
- **FALLBACK:** Indicates whether the custom metrics autoscaler is able to get metrics from the source
 - If **False**, the custom metrics autoscaler is getting metrics.
 - If **True**, the custom metrics autoscaler is getting metrics because there are no metrics or there is a problem in one or more of the objects you created.

3.6.2. Adding a custom metrics autoscaler to a job

You can create a custom metrics autoscaler for any **Job** object.



IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

Prerequisites

- The Custom Metrics Autoscaler Operator must be installed.

Procedure

1. Create a YAML file similar to the following:

```

kind: ScaledJob
apiVersion: keda.sh/v1alpha1
metadata:
  name: scaledjob
  namespace: my-namespace
spec:
  failedJobsHistoryLimit: 5
  jobTargetRef:
    activeDeadlineSeconds: 600 ❶
    backoffLimit: 6 ❷
    parallelism: 1 ❸
    completions: 1 ❹
    template: ❺
      metadata:
        name: pi
      spec:
        containers:
          - name: pi
            image: perl
            command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
  maxReplicaCount: 100 ❻
  pollingInterval: 30 ❼
  successfulJobsHistoryLimit: 5 ❽
  failedJobsHistoryLimit: 5 ❾
  envSourceContainerName: ❿
  rolloutStrategy: gradual ❶❶
  scalingStrategy: ❶❷
    strategy: "custom"
    customScalingQueueLengthDeduction: 1
    customScalingRunningJobPercentage: "0.5"
    pendingPodConditions:
      - "Ready"
      - "PodScheduled"
      - "AnyOtherCustomPodCondition"
    multipleScalersCalculation : "max"
  triggers:
    - type: prometheus ❶❸
      metadata:
        serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
        namespace: kedatest
        metricName: http_requests_total
        threshold: '5'
        query: sum(rate(http_requests_total{job="test-app"}[1m]))
        authModes: "bearer"
      authenticationRef: ❶❹
        name: prom-cluster-triggerauthentication

```

- ❶ Specifies the maximum duration the job can run.
- ❷ Specifies the number of retries for a job. The default is 6.
- ❸ Optional: Specifies how many pod replicas a job should run in parallel; defaults to 1.
 - For non-parallel jobs, leave unset. When unset, the default is 1.

- 4 Optional: Specifies how many successful pod completions are needed to mark a job completed.
 - For non-parallel jobs, leave unset. When unset, the default is **1**.
 - For parallel jobs with a fixed completion count, specify the number of completions.
 - For parallel jobs with a work queue, leave unset. When unset the default is the value of the **parallelism** parameter.
- 5 Specifies the template for the pod the controller creates.
- 6 Optional: Specifies the maximum number of replicas when scaling up. The default is **100**.
- 7 Optional: Specifies the interval in seconds to check each trigger on. The default is **30**.
- 8 Optional: Specifies the number of successful finished jobs should be kept. The default is **100**.
- 9 Optional: Specifies how many failed jobs should be kept. The default is **100**.
- 10 Optional: Specifies the name of the container in the target resource, from which the custom autoscaler gets environment variables holding secrets and so forth. The default is **.spec.template.spec.containers[0]**.
- 11 Optional: Specifies whether existing jobs are terminated whenever a scaled job is being updated:
 - **default**: The autoscaler terminates an existing job if its associated scaled job is updated. The autoscaler recreates the job with the latest specs.
 - **gradual**: The autoscaler does not terminate an existing job if its associated scaled job is updated. The autoscaler creates new jobs with the latest specs.
- 12 Optional: Specifies a scaling strategy: **default**, **custom**, or **accurate**. The default is **default**. For more information, see the link in the "Additional resources" section that follows.
- 13 Specifies the trigger to use as the basis for scaling, as described in the "Understanding the custom metrics autoscaler triggers" section.
- 14 Optional: Specifies a trigger authentication or a cluster trigger authentication. For more information, see *Understanding the custom metrics autoscaler trigger authentication* in the *Additional resources* section.
 - Enter **TriggerAuthentication** to use a trigger authentication. This is the default.
 - Enter **ClusterTriggerAuthentication** to use a cluster trigger authentication.

2. Create the custom metrics autoscaler by running the following command:

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

Verification

- View the command output to verify that the custom metrics autoscaler was created:

```
$ oc get scaledjob <scaled_job_name>
```

Example output

```
NAME      MAX TRIGGERS  AUTHENTICATION  READY  ACTIVE  AGE
scaledjob 100 prometheus  prom-triggerauthentication  True   True    8s
```

Note the following fields in the output:

- **TRIGGERS:** Indicates the trigger, or scaler, that is being used.
- **AUTHENTICATION:** Indicates the name of any trigger authentication being used.
- **READY:** Indicates whether the scaled object is ready to start scaling:
 - If **True**, the scaled object is ready.
 - If **False**, the scaled object is not ready because of a problem in one or more of the objects you created.
- **ACTIVE:** Indicates whether scaling is taking place:
 - If **True**, scaling is taking place.
 - If **False**, scaling is not taking place because there are no metrics or there is a problem in one or more of the objects you created.

3.6.3. Additional resources

- [Understanding custom metrics autoscaler trigger authentications](#)

3.7. PAUSING THE CUSTOM METRICS AUTOSCALER FOR A SCALED OBJECT

You can pause and restart the autoscaling of a workload, as needed.

For example, you might want to pause autoscaling before performing cluster maintenance or to avoid resource starvation by removing non-mission-critical workloads.

3.7.1. Pausing a custom metrics autoscaler

You can pause the autoscaling of a scaled object by adding the **autoscaling.keda.sh/paused-replicas** annotation to the custom metrics autoscaler for that scaled object. The custom metrics autoscaler scales the replicas for that workload to the specified value and pauses autoscaling until the annotation is removed.

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "4"
# ...
```


Procedure

1. Use the following command to edit the **ScaledObject** CR for your workload:

```
$ oc edit ScaledObject scaledobject
```

2. Add the **autoscaling.keda.sh/paused-replicas** annotation with any value:

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "4" 1
  creationTimestamp: "2023-02-08T14:41:01Z"
  generation: 1
  name: scaledobject
  namespace: my-project
  resourceVersion: '65729'
  uid: f5aec682-acdf-4232-a783-58b5b82f5dd0
```

- 1 Specifies that the Custom Metrics Autoscaler Operator is to scale the replicas to the specified value and stop autoscaling.

3.7.2. Restarting the custom metrics autoscaler for a scaled object

You can restart a paused custom metrics autoscaler by removing the **autoscaling.keda.sh/paused-replicas** annotation for that **ScaledObject**.

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "4"
# ...
```

Procedure

1. Use the following command to edit the **ScaledObject** CR for your workload:

```
$ oc edit ScaledObject scaledobject
```

2. Remove the **autoscaling.keda.sh/paused-replicas** annotation.

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "4" 1
  creationTimestamp: "2023-02-08T14:41:01Z"
  generation: 1
  name: scaledobject
```

```
namespace: my-project
resourceVersion: '65729'
uid: f5aec682-acdf-4232-a783-58b5b82f5dd0
```

- 1 Remove this annotation to restart a paused custom metrics autoscaler.

3.8. GATHERING AUDIT LOGS

You can gather audit logs, which are a security-relevant chronological set of records documenting the sequence of activities that have affected the system by individual users, administrators, or other components of the system.

For example, audit logs can help you understand where an autoscaling request is coming from. This is key information when backends are getting overloaded by autoscaling requests made by user applications and you need to determine which is the troublesome application.

3.8.1. Configuring audit logging

You can configure auditing for the Custom Metrics Autoscaler Operator by editing the **KedaController** custom resource. The logs are sent to an audit log file on a volume that is secured by using a persistent volume claim in the **KedaController** CR.

Prerequisites

- The Custom Metrics Autoscaler Operator must be installed.

Procedure

- Edit the **KedaController** custom resource to add the **auditConfig** stanza:

```
kind: KedaController
apiVersion: keda.sh/v1alpha1
metadata:
  name: keda
  namespace: openshift-keda
spec:
  # ...
  metricsServer:
  # ...
  auditConfig:
    logFormat: "json" 1
    logOutputVolumeClaim: "pvc-audit-log" 2
    policy:
      rules: 3
      - level: Metadata
      omitStages: "RequestReceived" 4
      omitManagedFields: false 5
    lifetime: 6
    maxAge: "2"
    maxBackup: "1"
    maxSize: "50"
```

- 1 Specifies the output format of the audit log, either **legacy** or **json**.
- 2 Specifies an existing persistent volume claim for storing the log data. All requests coming to the API server are logged to this persistent volume claim. If you leave this field empty, the log data is sent to stdout.
- 3 Specifies which events should be recorded and what data they should include:
 - **None**: Do not log events.
 - **Metadata**: Log only the metadata for the request, such as user, timestamp, and so forth. Do not log the request text and the response text. This is the default.
 - **Request**: Log only the metadata and the request text but not the response text. This option does not apply for non-resource requests.
 - **RequestResponse**: Log event metadata, request text, and response text. This option does not apply for non-resource requests.
- 4 Specifies stages for which no event is created.
- 5 Specifies whether to omit the managed fields of the request and response bodies from being written to the API audit log, either **true** to omit the fields or **false** to include the fields.
- 6 Specifies the size and lifespan of the audit logs.
 - **maxAge**: The maximum number of days to retain audit log files, based on the timestamp encoded in their filename.
 - **maxBackup**: The maximum number of audit log files to retain. Set to **0** to retain all audit log files.
 - **maxSize**: The maximum size in megabytes of an audit log file before it gets rotated.

Verification

1. View the audit log file directly:
 - a. Obtain the name of the **keda-metrics-apiserver-*** pod:

```
oc get pod -n openshift-keda
```

Example output

```
NAME                                READY STATUS RESTARTS AGE
custom-metrics-autoscaler-operator-5cb44cd75d-9v4lv 1/1 Running 0 8m20s
keda-metrics-apiserver-65c7cc44fd-rrl4r             1/1 Running 0 2m55s
keda-operator-776cbb6768-zpj5b                      1/1 Running 0 2m55s
```

- b. View the log data by using a command similar to the following:

```
$ oc logs keda-metrics-apiserver-<hash>|grep -i metadata 1
```

- 1 Optional: You can use the **grep** command to specify the log level to display: **Metadata, Request, RequestResponse**.

For example:

```
$ oc logs keda-metrics-apiserver-65c7cc44fd-rrl4r|grep -i metadata
```

Example output

```
...
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"4c81d41b-3dab-4675-90ce-20b87ce24013","stage":"ResponseComplete","requestURI":"/healthz","verb":"get","user":{"username":"system:anonymous","groups":["system:unauthenticated"],"sourceIPs":["10.131.0.1"],"userAgent":"kube-probe/1.28","responseStatus":{"metadata":{},"code":200},"requestReceivedTimestamp":"2023-02-16T13:00:03.554567Z","stageTimestamp":"2023-02-16T13:00:03.555032Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":""}}}
...
```

2. Alternatively, you can view a specific log:

- a. Use a command similar to the following to log into the **keda-metrics-apiserver-*** pod:

```
$ oc rsh pod/keda-metrics-apiserver-<hash> -n openshift-keda
```

For example:

```
$ oc rsh pod/keda-metrics-apiserver-65c7cc44fd-rrl4r -n openshift-keda
```

- b. Change to the **/var/audit-policy/** directory:

```
sh-4.4$ cd /var/audit-policy/
```

- c. List the available logs:

```
sh-4.4$ ls
```

Example output

```
log-2023.02.17-14:50 policy.yaml
```

- d. View the log, as needed:

```
sh-4.4$ cat <log_name>/<pvc_name>|grep -i <log_level> 1
```

- 1 Optional: You can use the **grep** command to specify the log level to display: **Metadata, Request, RequestResponse**.

For example:

```
sh-4.4$ cat log-2023.02.17-14:50/pvc-audit-log|grep -i Request
```

Example output

```
...
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Request","auditID":"63e7f68c-04ec-4f4d-8749-bf1656572a41","stage":"ResponseComplete","requestURI":"/openapi/v2","verb":"get","user":{"username":"system:aggregator","groups":["system:authenticated"]},"sourceIPs":["10.128.0.1"],"responseStatus":{"metadata":{"code":304},"requestReceivedTimestamp":"2023-02-17T13:12:55.035478Z","stageTimestamp":"2023-02-17T13:12:55.038346Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"system:discovery\" of ClusterRole \"system:discovery\" to Group \"system:authenticated\""}}}
...
```

3.9. GATHERING DEBUGGING DATA

When opening a support case, it is helpful to provide debugging information about your cluster to Red Hat Support.

To help troubleshoot your issue, provide the following information:

- Data gathered using the **must-gather** tool.
- The unique cluster ID.

You can use the **must-gather** tool to collect data about the Custom Metrics Autoscaler Operator and its components, including the following items:

- The **openshift-keda** namespace and its child objects.
- The Custom Metric Autoscaler Operator installation objects.
- The Custom Metric Autoscaler Operator CRD objects.

3.9.1. Gathering debugging data

The following command runs the **must-gather** tool for the Custom Metrics Autoscaler Operator:

```
$ oc adm must-gather --image="$(oc get packagemanifests openshift-custom-metrics-autoscaler-operator \
-n openshift-marketplace \
-o jsonpath='{.status.channels[?(@.name=="stable")].currentCSVDesc.annotations.containerImage}')
```

**NOTE**

The standard OpenShift Container Platform **must-gather** command, **oc adm must-gather**, does not collect Custom Metrics Autoscaler Operator data.

Prerequisites

- You are logged in to OpenShift Container Platform as a user with the **cluster-admin** role.
- The OpenShift Container Platform CLI (**oc**) installed.

Procedure

1. Navigate to the directory where you want to store the **must-gather** data.

**NOTE**

If your cluster is using a restricted network, you must take additional steps. If your mirror registry has a trusted CA, you must first add the trusted CA to the cluster. For all clusters on restricted networks, you must import the default **must-gather** image as an image stream by running the following command.

```
$ oc import-image is/must-gather -n openshift
```

2. Perform one of the following:

- To get only the Custom Metrics Autoscaler Operator **must-gather** data, use the following command:

```
$ oc adm must-gather --image="$(oc get packagemanifests openshift-custom-metrics-autoscaler-operator \
-n openshift-marketplace \
-o jsonpath='{.status.channels[?(@.name=="stable")].currentCSVDesc.annotations.containerImage}')"
```

The custom image for the **must-gather** command is pulled directly from the Operator package manifests, so that it works on any cluster where the Custom Metric Autoscaler Operator is available.

- To gather the default **must-gather** data in addition to the Custom Metric Autoscaler Operator information:

- a. Use the following command to obtain the Custom Metrics Autoscaler Operator image and set it as an environment variable:

```
$ IMAGE="$(oc get packagemanifests openshift-custom-metrics-autoscaler-operator \
-n openshift-marketplace \
-o jsonpath='{.status.channels[?(@.name=="stable")].currentCSVDesc.annotations.containerImage}')"
```

- b. Use the **oc adm must-gather** with the Custom Metrics Autoscaler Operator image:

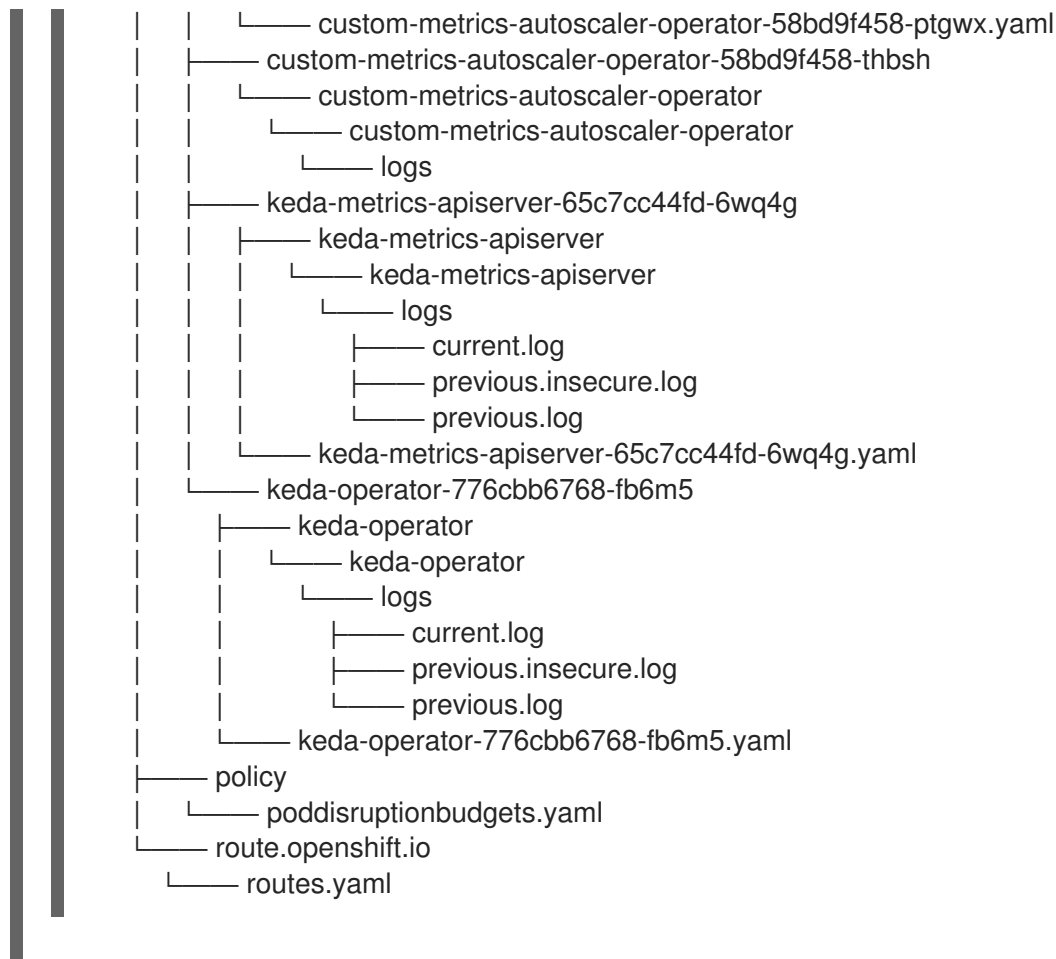
```
$ oc adm must-gather --image-stream=openshift/must-gather --image=${IMAGE}
```

Example 3.1. Example must-gather output for the Custom Metric Autoscaler

```

└── openshift-keda
    ├── apps
    │   ├── daemonsets.yaml
    │   ├── deployments.yaml
    │   ├── replicasetsets.yaml
    │   └── statefulsets.yaml
    ├── apps.openshift.io
    │   └── deploymentconfigs.yaml
    ├── autoscaling
    │   └── horizontalpodautoscalers.yaml
    ├── batch
    │   ├── cronjobs.yaml
    │   └── jobs.yaml
    ├── build.openshift.io
    │   ├── buildconfigs.yaml
    │   └── builds.yaml
    ├── core
    │   ├── configmaps.yaml
    │   ├── endpoints.yaml
    │   ├── events.yaml
    │   ├── persistentvolumeclaims.yaml
    │   ├── pods.yaml
    │   ├── replicationcontrollers.yaml
    │   ├── secrets.yaml
    │   └── services.yaml
    ├── discovery.k8s.io
    │   └── endpointslices.yaml
    ├── image.openshift.io
    │   └── imagestreams.yaml
    ├── k8s.ovn.org
    │   ├── egressfirewalls.yaml
    │   └── egressqoses.yaml
    ├── keda.sh
    │   ├── kedacontrollers
    │   │   └── keda.yaml
    │   ├── scaledobjects
    │   │   └── example-scaledobject.yaml
    │   └── triggerauthentications
    │       └── example-triggerauthentication.yaml
    ├── monitoring.coreos.com
    │   └── servicemonitors.yaml
    ├── networking.k8s.io
    │   └── networkpolicies.yaml
    ├── openshift-keda.yaml
    ├── pods
    │   ├── custom-metrics-autoscaler-operator-58bd9f458-ptgwx
    │   │   ├── custom-metrics-autoscaler-operator
    │   │   │   └── custom-metrics-autoscaler-operator
    │   │   │       └── logs
    │   │   │           ├── current.log
    │   │   │           ├── previous.insecure.log
    │   │   │           └── previous.log

```



3. Create a compressed file from the **must-gather** directory that was created in your working directory. For example, on a computer that uses a Linux operating system, run the following command:

```
$ tar cvaf must-gather.tar.gz must-gather-local.5421342344627712289/ 1
```

- 1 Replace **must-gather-local.5421342344627712289/** with the actual directory name.

4. Attach the compressed file to your support case on the [Red Hat Customer Portal](#).

3.10. VIEWING OPERATOR METRICS

The Custom Metrics Autoscaler Operator exposes ready-to-use metrics that it pulls from the on-cluster monitoring component. You can query the metrics by using the Prometheus Query Language (PromQL) to analyze and diagnose issues. All metrics are reset when the controller pod restarts.

3.10.1. Accessing performance metrics

You can access the metrics and run queries by using the OpenShift Container Platform web console.

Procedure

1. Select the **Administrator** perspective in the OpenShift Container Platform web console.
2. Select **Observe** → **Metrics**.

3. To create a custom query, add your PromQL query to the **Expression** field.
4. To add multiple queries, select **Add Query**.

3.10.1.1. Provided Operator metrics

The Custom Metrics Autoscaler Operator exposes the following metrics, which you can view by using the OpenShift Container Platform web console.

Table 3.1. Custom Metric Autoscaler Operator metrics

Metric name	Description
keda_scaler_activity	Whether the particular scaler is active or inactive. A value of 1 indicates the scaler is active; a value of 0 indicates the scaler is inactive.
keda_scaler_metrics_value	The current value for each scaler's metric, which is used by the Horizontal Pod Autoscaler (HPA) in computing the target average.
keda_scaler_metrics_latency	The latency of retrieving the current metric from each scaler.
keda_scaler_errors	The number of errors that have occurred for each scaler.
keda_scaler_errors_total	The total number of errors encountered for all scalers.
keda_scaled_object_errors	The number of errors that have occurred for each scaled object.
keda_resource_totals	The total number of Custom Metrics Autoscaler custom resources in each namespace for each custom resource type.
keda_trigger_totals	The total number of triggers by trigger type.

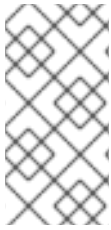
Custom Metrics Autoscaler Admission webhook metrics

The Custom Metrics Autoscaler Admission webhook also exposes the following Prometheus metrics.

Metric name	Description
keda_scaled_object_validation_total	The number of scaled object validations.
keda_scaled_object_validation_errors	The number of validation errors.

3.11. REMOVING THE CUSTOM METRICS AUTOSCALER OPERATOR

You can remove the custom metrics autoscaler from your OpenShift Container Platform cluster. After removing the Custom Metrics Autoscaler Operator, remove other components associated with the Operator to avoid potential issues.



NOTE

Delete the **KedaController** custom resource (CR) first. If you do not delete the **KedaController** CR, OpenShift Container Platform can hang when you delete the **openshift-keda** project. If you delete the Custom Metrics Autoscaler Operator before deleting the CR, you are not able to delete the CR.

3.11.1. Uninstalling the Custom Metrics Autoscaler Operator

Use the following procedure to remove the custom metrics autoscaler from your OpenShift Container Platform cluster.

Prerequisites

- The Custom Metrics Autoscaler Operator must be installed.

Procedure

1. In the OpenShift Container Platform web console, click **Ecosystem → Installed Operators**.
2. Switch to the **openshift-keda** project.
3. Remove the **KedaController** custom resource.
 - a. Find the **CustomMetricsAutoscaler** Operator and click the **KedaController** tab.
 - b. Find the custom resource, and then click **Delete KedaController**.
 - c. Click **Uninstall**.
4. Remove the Custom Metrics Autoscaler Operator:
 - a. Click **Ecosystem → Installed Operators**.
 - b. Find the **CustomMetricsAutoscaler** Operator and click the Options menu  and select **Uninstall Operator**.
 - c. Click **Uninstall**.
5. Optional: Use the OpenShift CLI to remove the custom metrics autoscaler components:
 - a. Delete the custom metrics autoscaler CRDs:
 - **clustertriggerauthentications.keda.sh**
 - **kedacontrollers.keda.sh**
 - **scaledjobs.keda.sh**
 - **scaledobjects.keda.sh**

- **triggerauthentications.keda.sh**

```
$ oc delete crd clustertriggerauthentications.keda.sh kedacontrollers.keda.sh  
scaledjobs.keda.sh scaledobjects.keda.sh triggerauthentications.keda.sh
```

Deleting the CRDs removes the associated roles, cluster roles, and role bindings. However, there might be a few cluster roles that must be manually deleted.

- b. List any custom metrics autoscaler cluster roles:

```
$ oc get clusterrole | grep keda.sh
```

- c. Delete the listed custom metrics autoscaler cluster roles. For example:

```
$ oc delete clusterrole.keda.sh-v1alpha1-admin
```

- d. List any custom metrics autoscaler cluster role bindings:

```
$ oc get clusterrolebinding | grep keda.sh
```

- e. Delete the listed custom metrics autoscaler cluster role bindings. For example:

```
$ oc delete clusterrolebinding.keda.sh-v1alpha1-admin
```

6. Delete the custom metrics autoscaler project:

```
$ oc delete project openshift-keda
```

7. Delete the Cluster Metric Autoscaler Operator:

```
$ oc delete operator/openshift-custom-metrics-autoscaler-operator.openshift-keda
```

CHAPTER 4. CONTROLLING POD PLACEMENT ONTO NODES (SCHEDULING)

4.1. CONTROLLING POD PLACEMENT USING THE SCHEDULER

Pod scheduling is an internal process that determines placement of new pods onto nodes within the cluster.

The scheduler code has a clean separation that watches new pods as they get created and identifies the most suitable node to host them. It then creates bindings (pod to node bindings) for the pods using the master API.

Default pod scheduling

OpenShift Container Platform comes with a default scheduler that serves the needs of most users. The default scheduler uses both inherent and customization tools to determine the best fit for a pod.

Advanced pod scheduling

In situations where you might want more control over where new pods are placed, the OpenShift Container Platform advanced scheduling features allow you to configure a pod so that the pod is required or has a preference to run on a particular node or alongside a specific pod.

You can control pod placement by using the following scheduling features:

- [Scheduler profiles](#)
- [Pod affinity and anti-affinity rules](#)
- [Node affinity](#)
- [Node selectors](#)
- [Taints and tolerations](#)
- [Node overcommitment](#)

4.1.1. About the default scheduler

The default OpenShift Container Platform pod scheduler is responsible for determining the placement of new pods onto nodes within the cluster. It reads data from the pod and finds a node that is a good fit based on configured profiles. It is completely independent and exists as a standalone solution. It does not modify the pod; it creates a binding for the pod that ties the pod to the particular node.

4.1.1.1. Understanding default scheduling

The existing generic scheduler is the default platform-provided scheduler *engine* that selects a node to host the pod in a three-step operation:

Filters the nodes

The available nodes are filtered based on the constraints or requirements specified. This is done by running each node through the list of filter functions called *predicates*, or *filters*.

Prioritizes the filtered list of nodes

This is achieved by passing each node through a series of *priority*, or *scoring*, functions that assign it a score between 0 - 10, with 0 indicating a bad fit and 10 indicating a good fit to host the pod. The

scheduler configuration can also take in a simple *weight* (positive numeric value) for each scoring function. The node score provided by each scoring function is multiplied by the weight (default weight for most scores is 1) and then combined by adding the scores for each node provided by all the scores. This weight attribute can be used by administrators to give higher importance to some scores.

Selects the best fit node

The nodes are sorted based on their scores and the node with the highest score is selected to host the pod. If multiple nodes have the same high score, then one of them is selected at random.

4.1.2. Scheduler use cases

One of the important use cases for scheduling within OpenShift Container Platform is to support flexible affinity and anti-affinity policies.

4.1.2.1. Infrastructure topological levels

Administrators can define multiple topological levels for their infrastructure (nodes) by specifying labels on nodes. For example: **region=r1, zone=z1, rack=s1**.

These label names have no particular meaning and administrators are free to name their infrastructure levels anything, such as city/building/room. Also, administrators can define any number of levels for their infrastructure topology, with three levels usually being adequate (such as: **regions → zones → racks**). Administrators can specify affinity and anti-affinity rules at each of these levels in any combination.

4.1.2.2. Affinity

Administrators should be able to configure the scheduler to specify affinity at any topological level, or even at multiple levels. Affinity at a particular level indicates that all pods that belong to the same service are scheduled onto nodes that belong to the same level. This handles any latency requirements of applications by allowing administrators to ensure that peer pods do not end up being too geographically separated. If no node is available within the same affinity group to host the pod, then the pod is not scheduled.

If you need greater control over where the pods are scheduled, see [Controlling pod placement on nodes using node affinity rules](#) and [Placing pods relative to other pods using affinity and anti-affinity rules](#).

These advanced scheduling features allow administrators to specify which node a pod can be scheduled on and to force or reject scheduling relative to other pods.

4.1.2.3. Anti-affinity

Administrators should be able to configure the scheduler to specify anti-affinity at any topological level, or even at multiple levels. Anti-affinity (or 'spread') at a particular level indicates that all pods that belong to the same service are spread across nodes that belong to that level. This ensures that the application is well spread for high availability purposes. The scheduler tries to balance the service pods across all applicable nodes as evenly as possible.

If you need greater control over where the pods are scheduled, see [Controlling pod placement on nodes using node affinity rules](#) and [Placing pods relative to other pods using affinity and anti-affinity rules](#).

These advanced scheduling features allow administrators to specify which node a pod can be scheduled on and to force or reject scheduling relative to other pods.

4.2. SCHEDULING PODS USING A SCHEDULER PROFILE

You can configure OpenShift Container Platform to use a scheduling profile to schedule pods onto nodes within the cluster.

4.2.1. About scheduler profiles

You can specify a scheduler profile to control how pods are scheduled onto nodes.

The following scheduler profiles are available:

LowNodeUtilization

This profile attempts to spread pods evenly across nodes to get low resource usage per node. This profile provides the default scheduler behavior.

HighNodeUtilization

This profile attempts to place as many pods as possible on to as few nodes as possible. This minimizes node count and has high resource usage per node.



NOTE

Switching to the **HighNodeUtilization** scheduler profile will result in all pods of a **ReplicaSet** object being scheduled on the same node. This will add an increased risk for pod failure if the node fails.

NoScoring

This is a low-latency profile that strives for the quickest scheduling cycle by disabling all score plugins. This might sacrifice better scheduling decisions for faster ones.

4.2.2. Configuring a scheduler profile

You can configure the scheduler to use a scheduler profile.

Prerequisites

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

Procedure

1. Edit the **Scheduler** object:

```
$ oc edit scheduler cluster
```

2. Specify the profile to use in the **spec.profile** field:

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
#...
spec:
  mastersSchedulable: false
  profile: HighNodeUtilization 1
#...
```

- 1 Set to **LowNodeUtilization**, **HighNodeUtilization**, or **NoScoring**.

3. Save the file to apply the changes.

4.3. PLACING PODS RELATIVE TO OTHER PODS USING AFFINITY AND ANTI-AFFINITY RULES

Affinity is a property of pods that controls the nodes on which they prefer to be scheduled. Anti-affinity is a property of pods that prevents a pod from being scheduled on a node.

In OpenShift Container Platform, *pod affinity* and *pod anti-affinity* allow you to constrain which nodes your pod is eligible to be scheduled on based on the key-value labels on other pods.

4.3.1. Understanding pod affinity

Pod affinity and *pod anti-affinity* allow you to constrain which nodes your pod is eligible to be scheduled on based on the key/value labels on other pods.

- Pod affinity can tell the scheduler to locate a new pod on the same node as other pods if the label selector on the new pod matches the label on the current pod.
- Pod anti-affinity can prevent the scheduler from locating a new pod on the same node as pods with the same labels if the label selector on the new pod matches the label on the current pod.

For example, using affinity rules, you could spread or pack pods within a service or relative to pods in other services. Anti-affinity rules allow you to prevent pods of a particular service from scheduling on the same nodes as pods of another service that are known to interfere with the performance of the pods of the first service. Or, you could spread the pods of a service across nodes, availability zones, or availability sets to reduce correlated failures.



NOTE

A label selector might match pods with multiple pod deployments. Use unique combinations of labels when configuring anti-affinity rules to avoid matching pods.

There are two types of pod affinity rules: *required* and *preferred*.

Required rules **must** be met before a pod can be scheduled on a node. Preferred rules specify that, if the rule is met, the scheduler tries to enforce the rules, but does not guarantee enforcement.



NOTE

Depending on your pod priority and preemption settings, the scheduler might not be able to find an appropriate node for a pod without violating affinity requirements. If so, a pod might not be scheduled.

To prevent this situation, carefully configure pod affinity with equal-priority pods.

You configure pod affinity/anti-affinity through the **Pod** spec files. You can specify a required rule, a preferred rule, or both. If you specify both, the node must first meet the required rule, then attempts to meet the preferred rule.

The following example shows a **Pod** spec configured for pod affinity and anti-affinity.

In this example, the pod affinity rule indicates that the pod can schedule onto a node only if that node has at least one already-running pod with a label that has the key **security** and value **S1**. The pod anti-affinity rule says that the pod prefers to not schedule onto a node if that node is already running a pod with label having key **security** and value **S2**.

Sample Pod config file with pod affinity

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  affinity:
    podAffinity: ❶
      requiredDuringSchedulingIgnoredDuringExecution: ❷
        - labelSelector:
            matchExpressions:
              - key: security ❸
                operator: In ❹
                values:
                  - S1 ❺
            topologyKey: topology.kubernetes.io/zone
  containers:
    - name: with-pod-affinity
      image: docker.io/ocpqe/hello-pod
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

- ❶ Stanza to configure pod affinity.
- ❷ Defines a required rule.
- ❸ ❺ The key and value (label) that must be matched to apply the rule.
- ❹ The operator represents the relationship between the label on the existing pod and the set of values in the **matchExpression** parameters in the specification for the new pod. Can be **In**, **NotIn**, **Exists**, or **DoesNotExist**.

Sample Pod config file with pod anti-affinity

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-antiaffinity
spec:
  securityContext:
    runAsNonRoot: true
```



```

seccompProfile:
  type: RuntimeDefault
affinity:
  podAntiAffinity: ❶
    preferredDuringSchedulingIgnoredDuringExecution: ❷
      - weight: 100 ❸
        podAffinityTerm:
          labelSelector:
            matchExpressions:
              - key: security ❹
                operator: In ❺
                values:
                  - S2
          topologyKey: kubernetes.io/hostname
containers:
  - name: with-pod-affinity
    image: docker.io/ocpqe/hello-pod
  securityContext:
    allowPrivilegeEscalation: false
  capabilities:
    drop: [ALL]

```

- ❶ Stanza to configure pod anti-affinity.
- ❷ Defines a preferred rule.
- ❸ Specifies a weight for a preferred rule. The node with the highest weight is preferred.
- ❹ Description of the pod label that determines when the anti-affinity rule applies. Specify a key and value for the label.
- ❺ The operator represents the relationship between the label on the existing pod and the set of values in the **matchExpression** parameters in the specification for the new pod. Can be **In**, **NotIn**, **Exists**, or **DoesNotExist**.



NOTE

If labels on a node change at runtime such that the affinity rules on a pod are no longer met, the pod continues to run on the node.

4.3.2. Configuring a pod affinity rule

The following steps demonstrate a simple two-pod configuration that creates pod with a label and a pod that uses affinity to allow scheduling with that pod.



NOTE

You cannot add an affinity directly to a scheduled pod.

Procedure

1. Create a pod with a specific label in the pod spec:
 - a. Create a YAML file with the following content:

```

apiVersion: v1
kind: Pod
metadata:
  name: security-s1
  labels:
    security: S1
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: security-s1
    image: docker.io/ocpqe/hello-pod
    securityContext:
      runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault

```

- b. Create the pod.

```
$ oc create -f <pod-spec>.yaml
```

2. When creating other pods, configure the following parameters to add the affinity:

- a. Create a YAML file with the following content:

```

apiVersion: v1
kind: Pod
metadata:
  name: security-s1-east
  # ...
spec:
  affinity: ❶
  podAffinity:
    requiredDuringSchedulingIgnoredDuringExecution: ❷
    - labelSelector:
        matchExpressions:
        - key: security ❸
          values:
          - S1
        operator: In ❹
      topologyKey: topology.kubernetes.io/zone ❺
  # ...

```

- ❶ Adds a pod affinity.
- ❷ Configures the **requiredDuringSchedulingIgnoredDuringExecution** parameter or the **preferredDuringSchedulingIgnoredDuringExecution** parameter.
- ❸ Specifies the **key** and **values** that must be met. If you want the new pod to be scheduled with the other pod, use the same **key** and **values** parameters as the label on the first pod.

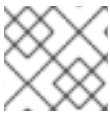
- 4 Specifies an **operator**. The operator can be **In**, **NotIn**, **Exists**, or **DoesNotExist**. For example, use the operator **In** to require the label to be in the node.
- 5 Specify a **topologyKey**, which is a prepopulated [Kubernetes label](#) that the system uses to denote such a topology domain.

b. Create the pod.

```
$ oc create -f <pod-spec>.yaml
```

4.3.3. Configuring a pod anti-affinity rule

The following steps demonstrate a simple two-pod configuration that creates pod with a label and a pod that uses an anti-affinity preferred rule to attempt to prevent scheduling with that pod.



NOTE

You cannot add an affinity directly to a scheduled pod.

Procedure

1. Create a pod with a specific label in the pod spec:
 - a. Create a YAML file with the following content:

```
apiVersion: v1
kind: Pod
metadata:
  name: security-s1
  labels:
    security: S1
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
  - name: security-s1
    image: docker.io/ocpqe/hello-pod
    securityContext:
      allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
```

b. Create the pod.

```
$ oc create -f <pod-spec>.yaml
```

2. When creating other pods, configure the following parameters:
 - a. Create a YAML file with the following content:

```
apiVersion: v1
```

```

kind: Pod
metadata:
  name: security-s2-east
# ...
spec:
# ...
  affinity: ❶
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution: ❷
      - weight: 100 ❸
        podAffinityTerm:
          labelSelector:
            matchExpressions:
              - key: security ❹
                values:
                  - S1
            operator: In ❺
          topologyKey: kubernetes.io/hostname ❻
# ...

```

- ❶ Adds a pod anti-affinity.
- ❷ Configures the **requiredDuringSchedulingIgnoredDuringExecution** parameter or the **preferredDuringSchedulingIgnoredDuringExecution** parameter.
- ❸ For a preferred rule, specifies a weight for the node, 1-100. The node that with highest weight is preferred.
- ❹ Specifies the **key** and **values** that must be met. If you want the new pod to not be scheduled with the other pod, use the same **key** and **values** parameters as the label on the first pod.
- ❺ Specifies an **operator**. The operator can be **In**, **NotIn**, **Exists**, or **DoesNotExist**. For example, use the operator **In** to require the label to be in the node.
- ❻ Specifies a **topologyKey**, which is a prepopulated [Kubernetes label](#) that the system uses to denote such a topology domain.

b. Create the pod.

```
$ oc create -f <pod-spec>.yaml
```

4.3.4. Sample pod affinity and anti-affinity rules

The following examples demonstrate pod affinity and pod anti-affinity.

4.3.4.1. Pod Affinity

The following example demonstrates pod affinity for pods with matching labels and label selectors.

- The pod **team4** has the label **team:4**.

```
apiVersion: v1
```

```

kind: Pod
metadata:
  name: team4
  labels:
    team: "4"
# ...
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: ocp
    image: docker.io/ocpqc/hello-pod
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...

```

- The pod **team4a** has the label selector **team:4** under **podAffinity**.

```

apiVersion: v1
kind: Pod
metadata:
  name: team4a
# ...
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: team
            operator: In
            values:
            - "4"
        topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-affinity
    image: docker.io/ocpqc/hello-pod
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...

```

- The **team4a** pod is scheduled on the same node as the **team4** pod.

4.3.4.2. Pod Anti-affinity

The following example demonstrates pod anti-affinity for pods with matching labels and label selectors.

- The pod **pod-s1** has the label **security:s1**.

```

apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
# ...
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...

```

- The pod **pod-s2** has the label selector **security:s1** under **podAntiAffinity**.

```

apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
# ...
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
              - key: security
                operator: In
                values:
                  - s1
            topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-antiaffinity
    image: docker.io/ocpqe/hello-pod
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...

```

- The pod **pod-s2** cannot be scheduled on the same node as **pod-s1**.

4.3.4.3. Pod Affinity with no Matching Labels

The following example demonstrates pod affinity for pods without matching labels and label selectors.

- The pod **pod-s1** has the label **security:s1**.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
# ...
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...
```

- The pod **pod-s2** has the label selector **security:s2**.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
# ...
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - s2
        topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-affinity
    image: docker.io/ocpqe/hello-pod
    securityContext:
```

```

    allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
# ...

```

- The pod **pod-s2** is not scheduled unless there is a node with a pod that has the **security:s2** label. If there is no other pod with that label, the new pod remains in a pending state:

Example output

```

NAME      READY   STATUS    RESTARTS   AGE   IP      NODE
pod-s2    0/1     Pending   0          32s   <none>

```

4.3.5. Using pod affinity and anti-affinity to control where an Operator is installed

By default, when you install an Operator, OpenShift Container Platform installs the Operator pod to one of your worker nodes randomly. However, there might be situations where you want that pod scheduled on a specific node or set of nodes.

The following examples describe situations where you might want to schedule an Operator pod to a specific node or set of nodes:

- If an Operator requires a particular platform, such as **amd64** or **arm64**
- If an Operator requires a particular operating system, such as Linux or Windows
- If you want Operators that work together scheduled on the same host or on hosts located on the same rack
- If you want Operators dispersed throughout the infrastructure to avoid downtime due to network or hardware issues

You can control where an Operator pod is installed by adding a pod affinity or anti-affinity to the Operator's **Subscription** object.

The following example shows how to use pod anti-affinity to prevent the installation the Custom Metrics Autoscaler Operator from any node that has pods with a specific label:

Pod affinity example that places the Operator pod on one or more specific nodes

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity:
      podAffinity: 1
        requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:

```



```

- key: app
  operator: In
  values:
  - test
topologyKey: kubernetes.io/hostname
#...

```

- 1 A pod affinity that places the Operator's pod on a node that has pods with the **app=test** label.

Pod anti-affinity example that prevents the Operator pod from one or more specific nodes

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
config:
  affinity:
    podAntiAffinity: 1
    requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
            - key: cpu
              operator: In
              values:
              - high
        topologyKey: kubernetes.io/hostname
#...

```

- 1 A pod anti-affinity that prevents the Operator's pod from being scheduled on a node that has pods with the **cpu=high** label.

Procedure

To control the placement of an Operator pod, complete the following steps:

1. Install the Operator as usual.
2. If needed, ensure that your nodes are labeled to properly respond to the affinity.
3. Edit the Operator **Subscription** object to add an affinity:

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators

```

```

sourceNamespace: operator-registries
config:
  affinity:
    podAntiAffinity: 1
      requiredDuringSchedulingIgnoredDuringExecution:
        podAffinityTerm:
          labelSelector:
            matchExpressions:
              - key: kubernetes.io/hostname
                operator: In
                values:
                  - ip-10-0-185-229.ec2.internal
          topologyKey: topology.kubernetes.io/zone
#...

```

- 1 Add a **podAffinity** or **podAntiAffinity**.

Verification

- To ensure that the pod is deployed on the specific node, run the following command:

```
$ oc get pods -o wide
```

Example output

NAME	READY	STATUS	RESTARTS	AGE	IP
custom-metrics-autoscaler-operator-5dcc45d656-bhshg	1/1	Running	0	50s	10.131.0.20
10.131.0.20	ip-10-0-185-229.ec2.internal	<none>	<none>		

4.4. CONTROLLING POD PLACEMENT ON NODES USING NODE AFFINITY RULES

Affinity is a property of pods that controls the nodes on which they prefer to be scheduled.

In OpenShift Container Platform node affinity is a set of rules used by the scheduler to determine where a pod can be placed. The rules are defined using custom labels on the nodes and label selectors specified in pods.

4.4.1. Understanding node affinity

Node affinity allows a pod to specify an affinity towards a group of nodes it can be placed on. The node does not have control over the placement.

For example, you could configure a pod to only run on a node with a specific CPU or in a specific availability zone.

There are two types of node affinity rules: *required* and *preferred*.

Required rules **must** be met before a pod can be scheduled on a node. Preferred rules specify that, if the rule is met, the scheduler tries to enforce the rules, but does not guarantee enforcement.

**NOTE**

If labels on a node change at runtime that results in an node affinity rule on a pod no longer being met, the pod continues to run on the node.

You configure node affinity through the **Pod** spec file. You can specify a required rule, a preferred rule, or both. If you specify both, the node must first meet the required rule, then attempts to meet the preferred rule.

The following example is a **Pod** spec with a rule that requires the pod be placed on a node with a label whose key is **e2e-az-NorthSouth** and whose value is either **e2e-az-North** or **e2e-az-South**:

Example pod configuration file with a node affinity required rule

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  affinity:
    nodeAffinity: 1
      requiredDuringSchedulingIgnoredDuringExecution: 2
        nodeSelectorTerms:
          - matchExpressions:
              - key: e2e-az-NorthSouth 3
                operator: In 4
                values:
                  - e2e-az-North 5
                  - e2e-az-South 6
        containers:
          - name: with-node-affinity
            image: docker.io/ocpqe/hello-pod
            securityContext:
              allowPrivilegeEscalation: false
            capabilities:
              drop: [ALL]
# ...
```

- 1 The stanza to configure node affinity.
- 2 Defines a required rule.
- 3 5 6 The key/value pair (label) that must be matched to apply the rule.
- 4 The operator represents the relationship between the label on the node and the set of values in the **matchExpression** parameters in the **Pod** spec. This value can be **In**, **NotIn**, **Exists**, or **DoesNotExist**, **Lt**, or **Gt**.

The following example is a node specification with a preferred rule that a node with a label whose key is **e2e-az-EastWest** and whose value is either **e2e-az-East** or **e2e-az-West** is preferred for the pod:

Example pod configuration file with a node affinity preferred rule

```

apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  affinity:
    nodeAffinity: ❶
      preferredDuringSchedulingIgnoredDuringExecution: ❷
        - weight: 1 ❸
          preference:
            matchExpressions:
              - key: e2e-az-EastWest ❹
                operator: In ❺
                values:
                  - e2e-az-East ❻
                  - e2e-az-West ❼
  containers:
    - name: with-node-affinity
      image: docker.io/ocpqe/hello-pod
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
# ...

```

- ❶ The stanza to configure node affinity.
- ❷ Defines a preferred rule.
- ❸ Specifies a weight for a preferred rule. The node with highest weight is preferred.
- ❹❻❼ The key/value pair (label) that must be matched to apply the rule.
- ❺ The operator represents the relationship between the label on the node and the set of values in the **matchExpression** parameters in the **Pod** spec. This value can be **In**, **NotIn**, **Exists**, or **DoesNotExist**, **Lt**, or **Gt**.

There is no explicit *node anti-affinity* concept, but using the **NotIn** or **DoesNotExist** operator replicates that behavior.



NOTE

If you are using node affinity and node selectors in the same pod configuration, note the following:

- If you configure both **nodeSelector** and **nodeAffinity**, both conditions must be satisfied for the pod to be scheduled onto a candidate node.
- If you specify multiple **nodeSelectorTerms** associated with **nodeAffinity** types, then the pod can be scheduled onto a node if one of the **nodeSelectorTerms** is satisfied.
- If you specify multiple **matchExpressions** associated with **nodeSelectorTerms**, then the pod can be scheduled onto a node only if all **matchExpressions** are satisfied.

4.4.2. Configuring a required node affinity rule

Required rules **must** be met before a pod can be scheduled on a node.

Procedure

The following steps demonstrate a simple configuration that creates a node and a pod that the scheduler is required to place on the node.

1. Add a label to a node using the **oc label node** command:

```
$ oc label node node1 e2e-az-name=e2e-az1
```

TIP

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

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    e2e-az-name: e2e-az1
#...
```

2. Create a pod with a specific label in the pod spec:
 - a. Create a YAML file with the following content:



NOTE

You cannot add an affinity directly to a scheduled pod.

Example output

```
apiVersion: v1
kind: Pod
```

```

metadata:
  name: s1
spec:
  affinity: ❶
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution: ❷
        nodeSelectorTerms:
          - matchExpressions:
              - key: e2e-az-name ❸
                values:
                  - e2e-az1
                  - e2e-az2
            operator: In ❹
#...

```

- ❶ Adds a pod affinity.
- ❷ Configures the **requiredDuringSchedulingIgnoredDuringExecution** parameter.
- ❸ Specifies the **key** and **values** that must be met. If you want the new pod to be scheduled on the node you edited, use the same **key** and **values** parameters as the label in the node.
- ❹ Specifies an **operator**. The operator can be **In**, **NotIn**, **Exists**, or **DoesNotExist**. For example, use the operator **In** to require the label to be in the node.

b. Create the pod:

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

4.4.3. Configuring a preferred node affinity rule

Preferred rules specify that, if the rule is met, the scheduler tries to enforce the rules, but does not guarantee enforcement.

Procedure

The following steps demonstrate a simple configuration that creates a node and a pod that the scheduler tries to place on the node.

1. Add a label to a node using the **oc label node** command:

```
$ oc label node node1 e2e-az-name=e2e-az3
```

2. Create a pod with a specific label:

a. Create a YAML file with the following content:



NOTE

You cannot add an affinity directly to a scheduled pod.

```
apiVersion: v1
```

```

kind: Pod
metadata:
  name: s1
spec:
  affinity: ❶
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution: ❷
      - weight: ❸
        preference:
          matchExpressions:
            - key: e2e-az-name ❹
              values:
                - e2e-az3
          operator: In ❺
#...

```

- ❶ Adds a pod affinity.
- ❷ Configures the **preferredDuringSchedulingIgnoredDuringExecution** parameter.
- ❸ Specifies a weight for the node, as a number 1-100. The node with highest weight is preferred.
- ❹ Specifies the **key** and **values** that must be met. If you want the new pod to be scheduled on the node you edited, use the same **key** and **values** parameters as the label in the node.
- ❺ Specifies an **operator**. The operator can be **In**, **NotIn**, **Exists**, or **DoesNotExist**. For example, use the operator **In** to require the label to be in the node.

b. Create the pod.

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

4.4.4. Sample node affinity rules

The following examples demonstrate node affinity.

4.4.4.1. Node affinity with matching labels

The following example demonstrates node affinity for a node and pod with matching labels:

- The Node1 node has the label **zone:us**:

```
$ oc label node node1 zone=us
```

TIP

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

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    zone: us
#...
```

- The pod-s1 pod has the **zone** and **us** key/value pair under a required node affinity rule:

```
$ cat pod-s1.yaml
```

Example output

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: "zone"
                operator: In
                values:
                  - us
#...
```

- The pod-s1 pod can be scheduled on Node1:

```
$ oc get pod -o wide
```

Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
pod-s1	1/1	Running	0	4m	IP1	node1

4.4.4.2. Node affinity with no matching labels

The following example demonstrates node affinity for a node and pod without matching labels:

- The Node1 node has the label **zone:emea**:

```
$ oc label node node1 zone=emea
```

TIP

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

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    zone: emea
#...
```

- The pod-s1 pod has the **zone** and **us** key/value pair under a required node affinity rule:

```
$ cat pod-s1.yaml
```

Example output

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: "zone"
                operator: In
                values:
                  - us
#...
```

- The pod-s1 pod cannot be scheduled on Node1:

```
$ oc describe pod pod-s1
```

Example output

```
...
Events:
  FirstSeen LastSeen Count From              SubObjectPath Type      Reason
  -----
  1m         33s      8    default-scheduler Warning    FailedScheduling  No nodes are
available that match all of the following predicates:: MatchNodeSelector (1).
```

4.4.5. Using node affinity to control where an Operator is installed

By default, when you install an Operator, OpenShift Container Platform installs the Operator pod to one of your worker nodes randomly. However, there might be situations where you want that pod scheduled on a specific node or set of nodes.

The following examples describe situations where you might want to schedule an Operator pod to a specific node or set of nodes:

- If an Operator requires a particular platform, such as **amd64** or **arm64**
- If an Operator requires a particular operating system, such as Linux or Windows
- If you want Operators that work together scheduled on the same host or on hosts located on the same rack
- If you want Operators dispersed throughout the infrastructure to avoid downtime due to network or hardware issues

You can control where an Operator pod is installed by adding a node affinity constraints to the Operator's **Subscription** object.

The following examples show how to use node affinity to install an instance of the Custom Metrics Autoscaler Operator to a specific node in the cluster:

Node affinity example that places the Operator pod on a specific node

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
config:
  affinity:
    nodeAffinity: 1
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
```

```

- matchExpressions:
- key: kubernetes.io/hostname
  operator: In
  values:
  - ip-10-0-163-94.us-west-2.compute.internal
#...

```

- 1 A node affinity that requires the Operator's pod to be scheduled on a node named **ip-10-0-163-94.us-west-2.compute.internal**.

Node affinity example that places the Operator pod on a node with a specific platform

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity:
      nodeAffinity: 1
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
        - key: kubernetes.io/arch
          operator: In
          values:
          - arm64
        - key: kubernetes.io/os
          operator: In
          values:
          - linux
#...

```

- 1 A node affinity that requires the Operator's pod to be scheduled on a node with the **kubernetes.io/arch=arm64** and **kubernetes.io/os=linux** labels.

Procedure

To control the placement of an Operator pod, complete the following steps:

1. Install the Operator as usual.
2. If needed, ensure that your nodes are labeled to properly respond to the affinity.
3. Edit the Operator **Subscription** object to add an affinity:

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator

```

```

namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity: ❶
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: kubernetes.io/hostname
                operator: In
                values:
                  - ip-10-0-185-229.ec2.internal
#...
```

- ❶ Add a **nodeAffinity**.

Verification

- To ensure that the pod is deployed on the specific node, run the following command:

```
$ oc get pods -o wide
```

Example output

NAME	READY	STATUS	RESTARTS	AGE	IP
NODE	NOMINATED NODE	READINESS GATES			
custom-metrics-autoscaler-operator-5dcc45d656-bhshg	1/1	Running	0	50s	
10.131.0.20	ip-10-0-185-229.ec2.internal	<none>	<none>		

4.4.6. Additional resources

- [Understanding how to update labels on nodes](#)

4.5. PLACING PODS ONTO OVERCOMMITTED NODES

In an *overcommitted* state, the sum of the container compute resource requests and limits exceeds the resources available on the system. Overcommitment might be desirable in development environments where a trade-off of guaranteed performance for capacity is acceptable.

Requests and limits enable administrators to allow and manage the overcommitment of resources on a node. The scheduler uses requests for scheduling your container and providing a minimum service guarantee. Limits constrain the amount of compute resource that may be consumed on your node.

4.5.1. Understanding overcommitment

Requests and limits enable administrators to allow and manage the overcommitment of resources on a node. The scheduler uses requests for scheduling your container and providing a minimum service guarantee. Limits constrain the amount of compute resource that may be consumed on your node.

OpenShift Container Platform administrators can control the level of overcommit and manage container density on nodes by configuring masters to override the ratio between request and limit set on developer containers. In conjunction with a per-project **LimitRange** object specifying limits and defaults, this adjusts the container limit and request to achieve the desired level of overcommit.



NOTE

That these overrides have no effect if no limits have been set on containers. Create a **LimitRange** object with default limits, per individual project, or in the project template, to ensure that the overrides apply.

After these overrides, the container limits and requests must still be validated by any **LimitRange** object in the project. It is possible, for example, for developers to specify a limit close to the minimum limit, and have the request then be overridden below the minimum limit, causing the pod to be forbidden. This unfortunate user experience should be addressed with future work, but for now, configure this capability and **LimitRange** objects with caution.

4.5.2. 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
```

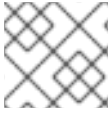
Example output

```
#...
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

4.6. CONTROLLING POD PLACEMENT USING NODE TAINTS

Taints and tolerations allow the node to control which pods should (or should not) be scheduled on them.

4.6.1. 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 4.1. Taint and toleration components

Parameter	Description						
key	The key is any string, up to 253 characters. The key must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores.						
value	The value is any string, up to 63 characters. The value must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores.						
effect	The effect is one of the following: <table> <tr> <td>NoSchedule ^[1]</td><td> - New pods that do not match the taint are not scheduled onto that node. - Existing pods on the node remain. </td></tr> <tr> <td>PreferNoSchedule</td><td> - New pods that do not match the taint might be scheduled onto that node, but the scheduler tries not to. - Existing pods on the node remain. </td></tr> <tr> <td>NoExecute</td><td> - New pods that do not match the taint cannot be scheduled onto that node. - Existing pods on the node that do not have a matching toleration are removed. </td></tr> </table>	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.						

1. If you add a **NoSchedule** taint to a control plane node, the node must have the **node-role.kubernetes.io/master=:NoSchedule** taint, which is added by default.

For example:

```
apiVersion: v1
kind: Node
metadata:
  annotations:
```

```

machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-f76d1-v8jxv-master-0
machineconfiguration.openshift.io/currentConfig: rendered-master-
cdc1ab7da414629332cc4c3926e6e59c
name: my-node
#...
spec:
  taints:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
#...

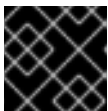
```

A toleration matches a taint:

- If the **operator** parameter is set to **Equal**:
 - the **key** parameters are the same;
 - the **value** parameters are the same;
 - the **effect** parameters are the same.
- If the **operator** parameter is set to **Exists**:
 - the **key** parameters are the same;
 - the **effect** parameters are the same.

The following taints are built into OpenShift Container Platform:

- **node.kubernetes.io/not-ready**: The node is not ready. This corresponds to the node condition **Ready=False**.
- **node.kubernetes.io/unreachable**: The node is unreachable from the node controller. This corresponds to the node condition **Ready=Unknown**.
- **node.kubernetes.io/memory-pressure**: The node has memory pressure issues. This corresponds to the node condition **MemoryPressure=True**.
- **node.kubernetes.io/disk-pressure**: The node has disk pressure issues. This corresponds to the node condition **DiskPressure=True**.
- **node.kubernetes.io/network-unavailable**: The node network is unavailable.
- **node.kubernetes.io/unschedulable**: The node is unschedulable.
- **node.cloudprovider.kubernetes.io/uninitialized**: When the node controller is started with an external cloud provider, this taint is set on a node to mark it as unusable. After a controller from the cloud-controller-manager initializes this node, the kubelet removes this taint.
- **node.kubernetes.io/pid-pressure**: The node has pid pressure. This corresponds to the node condition **PIDPressure=True**.



IMPORTANT

OpenShift Container Platform does not set a default pid.available **evictionHard**.

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

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

Example output

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
    tolerationSeconds: 3600
#...
```

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

4.6.1.2. Understanding how to use multiple taints

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

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

For example:

- Add the following taints to the node:

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

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

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

- The pod has the following tolerations:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
    - key: "key1"
      operator: "Equal"
      value: "value1"
      effect: "NoSchedule"
    - key: "key1"
      operator: "Equal"
      value: "value1"
      effect: "NoExecute"
#...
```

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

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

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

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

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

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

You can also add arbitrary tolerations to daemon sets.



NOTE

The control plane also adds the **node.kubernetes.io/memory-pressure** toleration on pods that have a QoS class. This is because Kubernetes manages pods in the **Guaranteed** or **Burstable** QoS classes. The new **BestEffort** pods do not get scheduled onto the affected node.

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

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

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

The **tolerationSeconds** parameter allows you to specify how long a pod stays bound to a node that has a node condition. If the condition still exists after the **tolerationSeconds** period, the taint remains on the node and the pods with a matching toleration are evicted. If the condition clears before the **tolerationSeconds** period, pods with matching tolerations are not removed.

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



NOTE

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

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

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

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

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: node.kubernetes.io/not-ready
    operator: Exists
    effect: NoExecute
```

```
tolerationSeconds: 300 1
- key: node.kubernetes.io/unreachable
  operator: Exists
  effect: NoExecute
  tolerationSeconds: 300
#...
```

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

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

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

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

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

4.6.1.5. Tolerating all taints

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

Pod spec for tolerating all taints

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - operator: "Exists"
#...
```

4.6.2. 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**.

4.6.2.1. 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" 1
      value: "value1"
      operator: "Equal"
      effect: "NoExecute"
      tolerationSeconds: 3600 2
#...
```

- 1** The toleration parameters, as described in the **Taint and toleration components** table.

- 2 The **tolerationSeconds** parameter specifies how long a pod is bound to a node before being evicted.

For example:

Sample pod configuration file with **Exists** operator

```
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.

4.6.2.2. 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.

4.6.2.3. Creating a project with a node selector and toleration

You can create a project that uses a node selector and toleration, which are set as annotations, to control the placement of pods onto specific nodes. Any subsequent resources created in the project are then scheduled on nodes that have a taint matching the toleration.

Prerequisites

- A label for node selection has been added to one or more nodes by using a compute machine set or editing the node directly.
- A taint has been added to one or more nodes by using a compute machine set or editing the node directly.

Procedure

1. Create a **Project** resource definition, specifying a node selector and toleration in the **metadata.annotations** section:

Example project.yaml file

```
kind: Project
apiVersion: project.openshift.io/v1
metadata:
  name: <project_name> ❶
  annotations:
    openshift.io/node-selector: '<label>' ❷
    scheduler.alpha.kubernetes.io/defaultTolerations: >-
      [{"operator": "Exists", "effect": "NoSchedule", "key":
        "<key_name>"}] ❸
  ]
```

- ❶ The project name.
- ❷ The default node selector label.

- 3 The toleration parameters, as described in the **Taint and toleration components** table. This example uses the **NoSchedule** effect, which allows existing pods on the node to

2. Use the **oc apply** command to create the project:

```
$ oc apply -f project.yaml
```

Any subsequent resources created in the **<project_name>** namespace should now be scheduled on the specified nodes.

Additional resources

- Adding taints and tolerations [manually to nodes](#) or [with compute machine sets](#)
- [Creating project-wide node selectors](#)
- [Pod placement of Operator workloads](#)

4.6.2.4. 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
#...
```

4.6.3. 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
#...

```

4.7. PLACING PODS ON SPECIFIC NODES USING NODE SELECTORS

A *node selector* specifies a map of key/value pairs that are defined using custom labels on nodes and selectors specified in pods.

For the pod to be eligible to run on a node, the pod must have the same key/value node selector as the label on the node.

4.7.1. About node selectors

You can use node selectors on pods and labels on nodes to control where the pod is scheduled. With node selectors, OpenShift Container Platform schedules the pods on nodes that contain matching labels.

You can use a node selector to place specific pods on specific nodes, cluster-wide node selectors to place new pods on specific nodes anywhere in the cluster, and project node selectors to place new pods in a project on specific nodes.

For example, as a cluster administrator, you can create an infrastructure where application developers can deploy pods only onto the nodes closest to their geographical location by including a node selector in every pod they create. In this example, the cluster consists of five data centers spread across two regions. In the U.S., label the nodes as **us-east**, **us-central**, or **us-west**. In the Asia-Pacific region (APAC), label the nodes as **apac-east** or **apac-west**. The developers can add a node selector to the pods they create to ensure the pods get scheduled on those nodes.

A pod is not scheduled if the **Pod** object contains a node selector, but no node has a matching label.

IMPORTANT

If you are using node selectors and node affinity in the same pod configuration, the following rules control pod placement onto nodes:

- If you configure both **nodeSelector** and **nodeAffinity**, both conditions must be satisfied for the pod to be scheduled onto a candidate node.
- If you specify multiple **nodeSelectorTerms** associated with **nodeAffinity** types, then the pod can be scheduled onto a node if one of the **nodeSelectorTerms** is satisfied.
- If you specify multiple **matchExpressions** associated with **nodeSelectorTerms**, then the pod can be scheduled onto a node only if all **matchExpressions** are satisfied.

Node selectors on specific pods and nodes

You can control which node a specific pod is scheduled on by using node selectors and labels.

To use node selectors and labels, first label the node to avoid pods being descheduled, then add the node selector to the pod.



NOTE

You cannot add a node selector directly to an existing scheduled pod. You must label the object that controls the pod, such as deployment config.

For example, the following **Node** object has the **region: east** label:

Sample Node object with a label

```
kind: Node
apiVersion: v1
metadata:
  name: ip-10-0-131-14.ec2.internal
  selfLink: /api/v1/nodes/ip-10-0-131-14.ec2.internal
  uid: 7bc2580a-8b8e-11e9-8e01-021ab4174c74
  resourceVersion: '478704'
  creationTimestamp: '2019-06-10T14:46:08Z'
  labels:
    kubernetes.io/os: linux
    topology.kubernetes.io/zone: us-east-1a
    node.openshift.io/os_version: '4.5'
    node-role.kubernetes.io/worker: "
    topology.kubernetes.io/region: us-east-1
    node.openshift.io/os_id: rhcos
    node.kubernetes.io/instance-type: m4.large
    kubernetes.io/hostname: ip-10-0-131-14
    kubernetes.io/arch: amd64
    region: east 1
    type: user-node
#...
```

1 Labels to match the pod node selector.

A pod has the **type: user-node,region: east** node selector:

Sample Pod object with node selectors

```
apiVersion: v1
kind: Pod
metadata:
  name: s1
#...
spec:
  nodeSelector: 1
    region: east
    type: user-node
#...
```

1 Node selectors to match the node label. The node must have a label for each node selector.

When you create the pod using the example pod spec, it can be scheduled on the example node.

Default cluster-wide node selectors

With default 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.

For example, the following **Scheduler** object has the default cluster-wide **region=east** and **type=user-node** node selectors:

Example Scheduler Operator Custom Resource

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

A node in that cluster has the **type=user-node,region=east** labels:

Example Node object

```
apiVersion: v1
kind: Node
metadata:
  name: ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
#...
labels:
  region: east
  type: user-node
#...
```

Example Pod object with a node selector

```
apiVersion: v1
kind: Pod
metadata:
  name: s1
#...
spec:
  nodeSelector:
    region: east
#...
```

When you create the pod using the example pod spec in the example cluster, the pod is created with the cluster-wide node selector and is scheduled on the labeled node:

Example pod list with the pod on the labeled node

```
NAME    READY  STATUS   RESTARTS  AGE  IP            NODE
NOMINATED NODE READINESS GATES
pod-s1  1/1    Running  0         20s  10.131.2.6    ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
<none>    <none>
```



NOTE

If the project where you create the pod has a project node selector, that selector takes preference over a cluster-wide node selector. Your pod is not created or scheduled if the pod does not have the project node selector.

Project node selectors

With project node selectors, when you create a pod in this project, OpenShift Container Platform adds the node selectors to the pod and schedules the pods on a node with matching labels. If there is a cluster-wide default node selector, a project node selector takes preference.

For example, the following project has the **region=east** node selector:

Example Namespace object

```
apiVersion: v1
kind: Namespace
metadata:
  name: east-region
  annotations:
    openshift.io/node-selector: "region=east"
#...
```

The following node has the **type=user-node,region=east** labels:

Example Node object

```
apiVersion: v1
kind: Node
metadata:
  name: ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
#...
labels:
  region: east
  type: user-node
#...
```

When you create the pod using the example pod spec in this example project, the pod is created with the project node selectors and is scheduled on the labeled node:

Example Pod object

```
apiVersion: v1
kind: Pod
metadata:
  namespace: east-region
#...
spec:
  nodeSelector:
    region: east
    type: user-node
#...
```

Example pod list with the pod on the labeled node

```

NAME      READY   STATUS    RESTARTS   AGE   IP            NODE
NOMINATED NODE READINESS GATES
pod-s1    1/1     Running   0          20s   10.131.2.6    ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
<none>    <none>

```

A pod in the project is not created or scheduled if the pod contains different node selectors. For example, if you deploy the following pod into the example project, it is not created:

Example Pod object with an invalid node selector

```

apiVersion: v1
kind: Pod
metadata:
  name: west-region
#...
spec:
  nodeSelector:
    region: west
#...

```

4.7.2. Using node selectors to control pod placement

You can use node selectors on pods and labels on nodes to control where the pod is scheduled. With node selectors, OpenShift Container Platform schedules the pods on nodes that contain matching labels.

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.

To add node selectors to an existing pod, add a node selector to the controlling object for that pod, such as a **ReplicaSet** object, **DaemonSet** object, **StatefulSet** object, **Deployment** object, or **DeploymentConfig** object. Any existing pods under that controlling object are recreated on a node with a matching label. If you are creating a new pod, you can add the node selector directly to the pod spec. If the pod does not have a controlling object, you must delete the pod, edit the pod spec, and recreate the pod.



NOTE

You cannot add a node selector directly to an existing scheduled pod.

Prerequisites

To add a node selector to existing pods, determine the controlling object for that pod. For example, the **router-default-66d5cf9464-m2g75** pod is controlled by the **router-default-66d5cf9464** replica set:

```
$ oc describe pod router-default-66d5cf9464-7pwkc
```

Example output


```

kind: Pod
apiVersion: v1
metadata:
# ...
Name:          router-default-66d5cf9464-7pwkc
Namespace:     openshift-ingress
# ...
Controlled By: ReplicaSet/router-default-66d5cf9464
# ...

```

The web console lists the controlling object under **ownerReferences** in the pod YAML:

```

apiVersion: v1
kind: Pod
metadata:
  name: router-default-66d5cf9464-7pwkc
# ...
ownerReferences:
- apiVersion: apps/v1
  kind: ReplicaSet
  name: router-default-66d5cf9464
  uid: d81dd094-da26-11e9-a48a-128e7edf0312
  controller: true
  blockOwnerDeletion: true
# ...

```

Procedure

1. Add labels to a node by using a compute machine set or editing the node directly:
 - Use a **MachineSet** object 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

```

For example:

```

$ oc patch MachineSet abc612-msrtw-worker-us-east-1c --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: xf2bd-infra-us-east-2a
  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
# ...
```

- 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 ip-10-0-142-25.ec2.internal 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: hello-node-6fbccf8d9
labels:
  type: "user-node"
  region: "east"
# ...
```

- b. Verify that the labels are added to the node:

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

Example output

```
NAME                                STATUS ROLES  AGE  VERSION
ip-10-0-142-25.ec2.internal Ready  worker  17m  v1.33.4
```

2. Add the matching node selector to a pod:

- To add a node selector to existing and future pods, add a node selector to the controlling object for the pods:

Example ReplicaSet object with labels

```
kind: ReplicaSet
apiVersion: apps/v1
metadata:
  name: hello-node-6fbccf8d9
# ...
spec:
# ...
template:
  metadata:
    creationTimestamp: null
    labels:
      ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
      pod-template-hash: 66d5cf9464
  spec:
    nodeSelector:
      kubernetes.io/os: linux
      node-role.kubernetes.io/worker: "
      type: user-node 1
# ...
```

- 1** Add the node selector.

- To add a node selector to a specific, new pod, add the selector to the **Pod** object directly:

Example Pod object with a node selector

```

apiVersion: v1
kind: Pod
metadata:
  name: hello-node-6fbccf8d9
# ...
spec:
  nodeSelector:
    region: east
    type: user-node
# ...

```



NOTE

You cannot add a node selector directly to an existing scheduled pod.

4.7.3. 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
```

4.7.4. Creating project-wide node selectors

You can use node selectors in a project together with labels on nodes to constrain all pods created in that project to the labeled nodes.

When you create a pod in this project, OpenShift Container Platform adds the node selectors to the pods in the project and schedules the pods on a node with matching labels in the project. If there is a cluster-wide default node selector, a project node selector takes preference.

You add node selectors to a project by editing the **Namespace** object to add the **openshift.io/node-selector** parameter. 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.

A pod is not scheduled if the **Pod** object contains a node selector, but no project has a matching node selector. When you create a pod from that spec, you receive an error similar to the following message:

Example error message

```
Error from server (Forbidden): error when creating "pod.yaml": pods "pod-4" is forbidden: pod node label selector conflicts with its project node label selector
```

**NOTE**

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

Procedure

To add a default project node selector:

1. Create a namespace or edit an existing namespace to add the **openshift.io/node-selector** parameter:

```
$ oc edit namespace <name>
```

Example output

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    openshift.io/node-selector: "type=user-node,region=east" 1
    openshift.io/description: ""
    openshift.io/display-name: ""
    openshift.io/requester: kube:admin
    openshift.io/sa.scc.mcs: s0:c30,c5
    openshift.io/sa.scc.supplemental-groups: 1000880000/10000
    openshift.io/sa.scc.uid-range: 1000880000/10000
  creationTimestamp: "2021-05-10T12:35:04Z"
  labels:
    kubernetes.io/metadata.name: demo
  name: demo
  resourceVersion: "145537"
  uid: 3f8786e3-1fcb-42e3-a0e3-e2ac54d15001
spec:
  finalizers:
    - kubernetes
```

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

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

- Use a **MachineSet** object 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
```

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 ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

Example output

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

- c. Redeploy the nodes associated with that compute machine set:
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,region=east
```

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 to add labels:

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

For example, to label a node:

```
$ oc label nodes ci-ln-l8nry52-f76d1-hl7m7-worker-c-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** object using the **oc get** command:

```
$ oc get nodes -l <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

Additional resources

- [Creating a project with a node selector and toleration](#)

4.8. CONTROLLING POD PLACEMENT BY USING POD TOPOLOGY SPREAD CONSTRAINTS

You can use pod topology spread constraints to provide fine-grained control over the placement of your pods across nodes, zones, regions, or other user-defined topology domains. Distributing pods across failure domains can help to achieve high availability and more efficient resource utilization.

4.8.1. Example use cases

- As an administrator, I want my workload to automatically scale between two to fifteen pods. I want to ensure that when there are only two pods, they are not placed on the same node, to avoid a single point of failure.
- As an administrator, I want to distribute my pods evenly across multiple infrastructure zones to reduce latency and network costs. I want to ensure that my cluster can self-heal if issues arise.

4.8.2. Important considerations

- Pods in an OpenShift Container Platform cluster are managed by *workload controllers* such as deployments, stateful sets, or daemon sets. These controllers define the desired state for a group of pods, including how they are distributed and scaled across the nodes in the cluster. You should set the same pod topology spread constraints on all pods in a group to avoid confusion. When using a workload controller, such as a deployment, the pod template typically handles this for you.
- Mixing different pod topology spread constraints can make OpenShift Container Platform behavior confusing and troubleshooting more difficult. You can avoid this by ensuring that all nodes in a topology domain are consistently labeled. OpenShift Container Platform automatically populates well-known labels, such as **kubernetes.io/hostname**. This helps avoid the need for manual labeling of nodes. These labels provide essential topology information, ensuring consistent node labeling across the cluster.
- Only pods within the same namespace are matched and grouped together when spreading due to a constraint.
- You can specify multiple pod topology spread constraints, but you must ensure that they do not conflict with each other. All pod topology spread constraints must be satisfied for a pod to be placed.

4.8.3. Understanding skew and maxSkew

Skew refers to the difference in the number of pods that match a specified label selector across different topology domains, such as zones or nodes.

The skew is calculated for each domain by taking the absolute difference between the number of pods in that domain and the number of pods in the domain with the lowest amount of pods scheduled. Setting a **maxSkew** value guides the scheduler to maintain a balanced pod distribution.

4.8.3.1. Example skew calculation

You have three zones (A, B, and C), and you want to distribute your pods evenly across these zones. If zone A has 5 pods, zone B has 3 pods, and zone C has 2 pods, to find the skew, you can subtract the number of pods in the domain with the lowest amount of pods scheduled from the number of pods currently in each zone. This means that the skew for zone A is 3, the skew for zone B is 1, and the skew for zone C is 0.

4.8.3.2. The maxSkew parameter

The **maxSkew** parameter defines the maximum allowable difference, or skew, in the number of pods between any two topology domains. If **maxSkew** is set to **1**, the number of pods in any topology domain should not differ by more than 1 from any other domain. If the skew exceeds **maxSkew**, the scheduler attempts to place new pods in a way that reduces the skew, adhering to the constraints.

Using the previous example skew calculation, the skew values exceed the default **maxSkew** value of **1**. The scheduler places new pods in zone B and zone C to reduce the skew and achieve a more balanced distribution, ensuring that no topology domain exceeds the skew of 1.

4.8.4. Example configurations for pod topology spread constraints

You can specify which pods to group together, which topology domains they are spread among, and the acceptable skew.

The following examples demonstrate pod topology spread constraint configurations.

Example to distribute pods that match the specified labels based on their zone

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    region: us-east
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  topologySpreadConstraints:
    - maxSkew: 1 1
      topologyKey: topology.kubernetes.io/zone 2
      whenUnsatisfiable: DoNotSchedule 3
      labelSelector: 4
        matchLabels:
          region: us-east 5
      matchLabelKeys:
        - my-pod-label 6
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

¹ The maximum difference in number of pods between any two topology domains. The default is **1**, and you cannot specify a value of **0**.

² The key of a node label. Nodes with this key and identical value are considered to be in the same topology.

- 3 How to handle a pod if it does not satisfy the spread constraint. The default is **DoNotSchedule**, which tells the scheduler not to schedule the pod. Set to **ScheduleAnyway** to still schedule the pod, but the scheduler prioritizes honoring the skew to not make the cluster more imbalanced.
- 4 Pods that match this label selector are counted and recognized as a group when spreading to satisfy the constraint. Be sure to specify a label selector, otherwise no pods can be matched.
- 5 Be sure that this **Pod** spec also sets its labels to match this label selector if you want it to be counted properly in the future.
- 6 A list of pod label keys to select which pods to calculate spreading over.

Example demonstrating a single pod topology spread constraint

```
kind: Pod
apiVersion: v1
metadata:
  name: my-pod
  labels:
    region: us-east
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: DoNotSchedule
    labelSelector:
      matchLabels:
        region: us-east
  containers:
  - image: "docker.io/ocpqe/hello-pod"
    name: hello-pod
    securityContext:
      allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
```

The previous example defines a **Pod** spec with a one pod topology spread constraint. It matches on pods labeled **region: us-east**, distributes among zones, specifies a skew of **1**, and does not schedule the pod if it does not meet these requirements.

Example demonstrating multiple pod topology spread constraints

```
kind: Pod
apiVersion: v1
metadata:
  name: my-pod-2
  labels:
    region: us-east
spec:
  securityContext:
```

```

runAsNonRoot: true
seccompProfile:
  type: RuntimeDefault
topologySpreadConstraints:
- maxSkew: 1
  topologyKey: node
  whenUnsatisfiable: DoNotSchedule
  labelSelector:
    matchLabels:
      region: us-east
- maxSkew: 1
  topologyKey: rack
  whenUnsatisfiable: DoNotSchedule
  labelSelector:
    matchLabels:
      region: us-east
containers:
- image: "docker.io/ocpqe/hello-pod"
  name: hello-pod
  securityContext:
    allowPrivilegeEscalation: false
  capabilities:
    drop: [ALL]

```

The previous example defines a **Pod** spec with two pod topology spread constraints. Both match on pods labeled **region: us-east**, specify a skew of **1**, and do not schedule the pod if it does not meet these requirements.

The first constraint distributes pods based on a user-defined label **node**, and the second constraint distributes pods based on a user-defined label **rack**. Both constraints must be met for the pod to be scheduled.

4.8.5. Additional resources

- [Understanding how to update labels on nodes](#)

4.9. DESCHEDULER

4.9.1. Descheduler overview

While the [scheduler](#) is used to determine the most suitable node to host a new pod, the descheduler can be used to evict a running pod so that the pod can be rescheduled onto a more suitable node.

4.9.1.1. About the descheduler

You can use the descheduler to evict pods based on specific strategies so that the pods are rescheduled onto more appropriate nodes.

You can benefit from descheduling running pods in situations such as the following:

- Nodes are underutilized or overutilized.
- Pod and node affinity requirements, such as taints or labels, have changed and the original scheduling decisions are no longer appropriate for certain nodes.

- Node failure requires pods to be moved.
- New nodes are added to clusters.
- Pods have been restarted too many times.



IMPORTANT

The descheduler does not schedule replacement of evicted pods. The scheduler automatically performs this task for the evicted pods.

When the descheduler decides to evict pods from a node, it employs the following general mechanism:

- Pods in the **openshift-*** and **kube-system** namespaces are never evicted.
- Critical pods with **priorityClassName** set to **system-cluster-critical** or **system-node-critical** are never evicted.
- Static, mirrored, or standalone pods that are not part of a replication controller, replica set, deployment, stateful set, or job are never evicted because these pods are not recreated.
- Pods associated with daemon sets are never evicted.
- Pods with local storage are never evicted.
- Best effort pods are evicted before burstable and guaranteed pods.
- All types of pods with the **descheduler.alpha.kubernetes.io/evict** annotation are eligible for eviction. This annotation is used to override checks that prevent eviction, and the user can select which pod is evicted. Users should know how and if the pod will be recreated.
- Pods subject to pod disruption budget (PDB) are not evicted if descheduling violates its pod disruption budget (PDB). The pods are evicted by using eviction subresource to handle PDB.

4.9.1.2. Descheduler profiles

Use descheduler profiles to enable specific eviction strategies that rebalance your cluster based on criteria such as pod lifecycle or node utilization.

The following descheduler profiles are available:

AffinityAndTaints

This profile evicts pods that violate inter-pod anti-affinity, node affinity, and node taints. It enables the following strategies:

- **RemovePodsViolatingInterPodAntiAffinity**: removes pods that are violating inter-pod anti-affinity.
- **RemovePodsViolatingNodeAffinity**: removes pods that are violating node affinity.
- **RemovePodsViolatingNodeTaints**: removes pods that are violating **NoSchedule** taints on nodes.
Pods with a node affinity type of **requiredDuringSchedulingIgnoredDuringExecution** are removed.

TopologyAndDuplicates

This profile evicts pods in an effort to evenly spread similar pods, or pods of the same topology domain, among nodes.

It enables the following strategies:

- **RemovePodsViolatingTopologySpreadConstraint:** finds unbalanced topology domains and tries to evict pods from larger ones when **DoNotSchedule** constraints are violated.
- **RemoveDuplicates:** ensures that there is only one pod associated with a replica set, replication controller, deployment, or job running on same node. If there are more, those duplicate pods are evicted for better pod distribution in a cluster.



WARNING

Do not enable **TopologyAndDuplicates** with any of the following profiles: **SoftTopologyAndDuplicates** or **CompactAndScale**. Enabling these profiles together results in a conflict.

LifecycleAndUtilization

This profile evicts long-running pods and balances resource usage between nodes.

It enables the following strategies:

- **RemovePodsHavingTooManyRestarts:** removes pods whose containers have been restarted too many times.
Pods where the sum of restarts over all containers (including Init Containers) is more than 100.
- **LowNodeUtilization:** finds nodes that are underutilized and evicts pods, if possible, from overutilized nodes in the hope that recreation of evicted pods will be scheduled on these underutilized nodes.
 - A node is considered underutilized if its usage is below 20% for all thresholds (CPU, memory, and number of pods).
 - A node is considered overutilized if its usage is above 50% for any of the thresholds (CPU, memory, and number of pods).

Optionally, you can adjust these underutilized/overutilized threshold percentages by setting the Technology Preview field **devLowNodeUtilizationThresholds** to one of the following values: **Low** for 10%/30%, **Medium** for 20%/50%, or **High** for 40%/70%. The default value is **Medium**.

- **PodLifeTime:** evicts pods that are too old.
By default, pods that are older than 24 hours are removed. You can customize the pod lifetime value.

**WARNING**

Do not enable **LifecycleAndUtilization** with any of the following profiles: **LongLifecycle** or **CompactAndScale**. Enabling these profiles together results in a conflict.

SoftTopologyAndDuplicates

This profile is the same as **TopologyAndDuplicates**, except that pods with soft topology constraints, such as **whenUnsatisfiable: ScheduleAnyway**, are also considered for eviction.

**WARNING**

Do not enable both **SoftTopologyAndDuplicates** and **TopologyAndDuplicates**. Enabling both results in a conflict.

EvictPodsWithLocalStorage

This profile allows pods with local storage to be eligible for eviction.

EvictPodsWithPVC

This profile allows pods with persistent volume claims to be eligible for eviction. If you are using **Kubernetes NFS Subdir External Provisioner**, you must add an excluded namespace for the namespace where the provisioner is installed.

CompactAndScale

This profile enables the **HighNodeUtilization** strategy, which attempts to evict pods from underutilized nodes to allow a workload to run on a smaller set of nodes. A node is considered underutilized if its usage is below 20% for all thresholds (CPU, memory, and number of pods). Optionally, you can adjust the underutilized percentage by setting the Technology Preview field **devHighNodeUtilizationThresholds** to one of the following values: **Minimal** for 10%, **Modest** for 20%, or **Moderate** for 30%. The default value is **Modest**.

**WARNING**

Do not enable **CompactAndScale** with any of the following profiles: **LifecycleAndUtilization**, **LongLifecycle**, or **TopologyAndDuplicates**. Enabling these profiles together results in a conflict.

KubeVirtRelieveAndMigrate

This profile is an enhanced version of the **LongLifeCycle** profile.

The **KubeVirtRelieveAndMigrate** profile evicts pods from high-cost nodes to reduce overall resource expenses and enable workload migration. It also periodically rebalances workloads to help maintain similar spare capacity across nodes, which supports better handling of sudden workload spikes. Nodes can experience the following costs:

- **Resource utilization:** Increased resource pressure raises the overhead for running applications.
- **Node maintenance:** A higher number of containers on a node increases resource consumption and maintenance costs.

The profile enables the **LowNodeUtilization** strategy with the **EvictionsInBackground** alpha feature. The profile also exposes the following customization fields:

- **devActualUtilizationProfile:** Enables load-aware descheduling.
- **devLowNodeUtilizationThresholds:** Sets experimental thresholds for the **LowNodeUtilization** strategy. Do not use this field with **devDeviationThresholds**.
- **devDeviationThresholds:** Treats nodes with below-average resource usage as underutilized to help redistribute workloads from overutilized nodes. Do not use this field with **devLowNodeUtilizationThresholds**. Supported values are: **Low** (10%:10%), **Medium** (20%:20%), **High** (30%:30%), **AsymmetricLow** (0%:10%), **AsymmetricMedium** (0%:20%), **AsymmetricHigh** (0%:30%).
- **devEnableSoftTainter:** Enables the soft-tainting component to dynamically apply or remove soft taints as scheduling hints.

Example configuration

```
apiVersion: operator.openshift.io/v1
kind: KubeDescheduler
metadata:
  name: cluster
  namespace: openshift-kube-descheduler-operator
spec:
  managementState: Managed
  deschedulingIntervalSeconds: 30
  mode: "Automatic"
  profiles:
    - KubeVirtRelieveAndMigrate
  profileCustomizations:
    devEnableSoftTainter: true
    devDeviationThresholds: AsymmetricLow
    devActualUtilizationProfile: PrometheusCPUCombined
```

The **KubeVirtRelieveAndMigrate** profile requires PSI metrics to be enabled on all worker nodes. You can enable this by applying the following **MachineConfig** custom resource (CR):

Example MachineConfig CR

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
```

```

labels:
  machineconfiguration.openshift.io/role: worker
name: 99-openshift-machineconfig-worker-psi-karg
spec:
  kernelArguments:
    - psi=1

```



NOTE

The name of the **MachineConfig** object is significant because machine configs are processed in lexicographical order. By default, a config that starts with **98-** disables PSI. To ensure that PSI is enabled, name your config with a higher prefix, such as **99-openshift-machineconfig-worker-psi-karg**.

You can use this profile with the **SoftTopologyAndDuplicates** profile to also rebalance pods based on soft topology constraints, which can be useful in hosted control plane environments.

LongLifecycle

This profile balances resource usage between nodes and enables the following strategies:

- **RemovePodsHavingTooManyRestarts:** removes pods whose containers have been restarted too many times and pods where the sum of restarts over all containers (including Init Containers) is more than 100. Restarting the VM guest operating system does not increase this count.
- **LowNodeUtilization:** evicts pods from overutilized nodes when there are any underutilized nodes. The destination node for the evicted pod will be determined by the scheduler.
 - A node is considered underutilized if its usage is below 20% for all thresholds (CPU, memory, and number of pods).
 - A node is considered overutilized if its usage is above 50% for any of the thresholds (CPU, memory, and number of pods).



WARNING

Do not enable **LongLifecycle** with any of the following profiles: **LifecycleAndUtilization** or **CompactAndScale**. Enabling these profiles together results in a conflict.

4.9.2. Kube Descheduler Operator release notes

Review the Kube Descheduler Operator release notes to track its development and learn what is new and changed with each release.

The Kube Descheduler Operator allows you to evict pods so that they can be rescheduled on more appropriate nodes.

For more information, see [About the descheduler](#).

4.9.2.1. Release notes for Kube Descheduler Operator 5.3.1

Review the release notes for Kube Descheduler Operator 5.3.1 to learn what is new and updated with this release.

Issued: 4 December 2025

The following advisory is available for the Kube Descheduler Operator 5.3.1:

- [RHBA-2025:22737](#)

4.9.2.1.1. New features and enhancements

- This release rebuilds the Kube Descheduler Operator to improve its image grade.

4.9.2.2. Release notes for Kube Descheduler Operator 5.3.0

Review the release notes for Kube Descheduler Operator 5.3.0 to learn what is new and updated with this release.

Issued: 29 October 2025

The following advisory is available for the Kube Descheduler Operator 5.3.0:

- [RHBA-2025:19249](#)

4.9.2.2.1. New features and enhancements

- The descheduler profile **DevKubeVirtRelieveAndMigrate** has been renamed to **KubeVirtRelieveAndMigrate** and is now generally available. The updated profile improves VM eviction stability during live migrations by enabling background evictions and reducing oscillatory behavior. This profile is only available for use with OpenShift Virtualization. For more information, see [Configuring descheduler evictions for virtual machines](#).
- This release of the Kube Descheduler Operator updates the Kubernetes version to 1.33.

4.9.2.3. Release notes for Kube Descheduler Operator 5.2.1

Review the release notes for Kube Descheduler Operator 5.2.1 to learn what is new and updated with this release.

Issued: 4 December 2025

The following advisory is available for the Kube Descheduler Operator 5.2.1:

- [RHBA-2025:22736](#)

4.9.2.3.1. New features and enhancements

- This release rebuilds the Kube Descheduler Operator to improve its image grade.

4.9.2.4. Release notes for Kube Descheduler Operator 5.2.0

Review the release notes for Kube Descheduler Operator 5.2.0 to learn what is new and updated with this release.

Issued: 9 July 2025

The following advisory is available for the Kube Descheduler Operator 5.2.0:

- [RHBA-2025:10724](#)

4.9.2.4.1. New features and enhancements

- This release of the Kube Descheduler Operator updates the Kubernetes version to 1.32.

4.9.2.4.2. Bug fixes

- This release of the Kube Descheduler Operator addresses several Common Vulnerabilities and Exposures (CVEs).

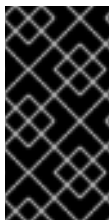
4.9.3. Evicting pods using the descheduler

You can run the descheduler in OpenShift Container Platform by installing the Kube Descheduler Operator and setting the required profiles and other customizations.

4.9.3.1. Installing the descheduler

The descheduler is not available by default. To enable the descheduler, you must install the Kube Descheduler Operator from the software catalog and enable one or more descheduler profiles.

By default, the descheduler runs in predictive mode, which means that it only simulates pod evictions. You must change the mode to automatic for the descheduler to perform the pod evictions.



IMPORTANT

If you have enabled hosted control planes in your cluster, set a custom priority threshold to lower the chance that pods in the hosted control plane namespaces are evicted. Set the priority threshold class name to **hypershift-control-plane**, because it has the lowest priority value (**100000000**) of the hosted control plane priority classes.

Prerequisites

- You are logged in to OpenShift Container Platform as a user with the **cluster-admin** role.
- Access to the OpenShift Container Platform web console.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Create the required namespace for the Kube Descheduler Operator.
 - a. Navigate to **Administration** → **Namespaces** and click **Create Namespace**.
 - b. Enter **openshift-kube-descheduler-operator** in the **Name** field, enter **openshift.io/cluster-monitoring=true** in the **Labels** field to enable descheduler metrics, and click **Create**.

3. Install the Kube Descheduler Operator.
 - a. Navigate to **Ecosystem → Software Catalog**.
 - b. Type **Kube Descheduler Operator** into the filter box.
 - c. Select the **Kube Descheduler Operator** and click **Install**.
 - d. On the **Install Operator** page, select **A specific namespace on the cluster**. Select **openshift-kube-descheduler-operator** from the drop-down menu.
 - e. Adjust the values for the **Update Channel** and **Approval Strategy** to the desired values.
 - f. Click **Install**.
4. Create a descheduler instance.
 - a. From the **Ecosystem → Installed Operators** page, click the **Kube Descheduler Operator**.
 - b. Select the **Kube Descheduler** tab and click **Create KubeDescheduler**.
 - c. Edit the settings as necessary.
 - i. To evict pods instead of simulating the evictions, change the **Mode** field to **Automatic**.
 - ii. Expand the **Profiles** section to select one or more profiles to enable. The **AffinityAndTaints** profile is enabled by default. Click **Add Profile** to select additional profiles.

**NOTE**

Do not enable both **TopologyAndDuplicates** and **SoftTopologyAndDuplicates**. Enabling both results in a conflict.

- iii. Optional: Expand the **Profile Customizations** section to set optional configurations for the descheduler.
 - Set a custom pod lifetime value for the **LifecycleAndUtilization** profile. Use the **podLifetime** field to set a numerical value and a valid unit (**s**, **m**, or **h**). The default pod lifetime is 24 hours (**24h**).
 - Set a custom priority threshold to consider pods for eviction only if their priority is lower than a specified priority level. Use the **thresholdPriority** field to set a numerical priority threshold or use the **thresholdPriorityClassName** field to specify a certain priority class name.

**NOTE**

Do not specify both **thresholdPriority** and **thresholdPriorityClassName** for the descheduler.

- Set specific namespaces to exclude or include from descheduler operations. Expand the **namespaces** field and add namespaces to the **excluded** or **included** list. You can only either set a list of namespaces to exclude or a list of namespaces to include. Note that protected namespaces (**openshift-***, **kube-system**, **hypershift**) are excluded by default.

- Experimental: Set thresholds for underutilization and overutilization for the **LowNodeUtilization** strategy. Use the **devLowNodeUtilizationThresholds** field to set one of the following values:
 - **Low**: 10% underutilized and 30% overutilized
 - **Medium**: 20% underutilized and 50% overutilized (Default)
 - **High**: 40% underutilized and 70% overutilized

**NOTE**

This setting is experimental and should not be used in a production environment.

- iv. Optional: Use the **Descheduling Interval Seconds** field to change the number of seconds between descheduler runs. The default is **3600** seconds.

- d. Click **Create**.

You can also configure the profiles and settings for the descheduler later using the OpenShift CLI (**oc**). If you did not adjust the profiles when creating the descheduler instance from the web console, the **AffinityAndTaints** profile is enabled by default.

4.9.3.2. Configuring descheduler profiles

To manage cluster pod eviction behavior, select which descheduler profiles to enable.

Prerequisites

- You are logged in to OpenShift Container Platform as a user with the **cluster-admin** role.

Procedure

1. Edit the **KubeDescheduler** object:

```
$ oc edit kubedeschedulers.operator.openshift.io cluster -n openshift-kube-descheduler-operator
```

2. Specify one or more profiles in the **spec.profiles** section.

```
apiVersion: operator.openshift.io/v1
kind: KubeDescheduler
metadata:
  name: cluster
  namespace: openshift-kube-descheduler-operator
spec:
  deschedulingIntervalSeconds: 3600
  logLevel: Normal
  managementState: Managed
  operatorLogLevel: Normal
  mode: Predictive
  profileCustomizations:
    namespaces:
```

```

    excluded:
      - my-namespace
    podLifetime: 48h
    thresholdPriorityClassName: my-priority-class-name
  evictionLimits:
    total: 20
  profiles:
    - AffinityAndTaints
    - TopologyAndDuplicates
    - LifecycleAndUtilization
    - EvictPodsWithLocalStorage
    - EvictPodsWithPVC

```

where:

spec.mode

Specifies the eviction mode. By default, the descheduler does not evict pods. To evict pods, set **mode** to **Automatic**.

spec.profileCustomizations.namespaces

Specifies a list of user-created namespaces to include or exclude from descheduler operations. Use **excluded** to set a list of namespaces to exclude or use **included** to set a list of namespaces to include. Note that protected namespaces (**openshift-***, **kube-system**, **hypershift**) are excluded by default. This value is optional.

spec.profileCustomizations.podLifetime

Specifies a custom pod lifetime value for the **LifecycleAndUtilization** profile. Valid units are **s**, **m**, or **h**. The default pod lifetime is 24 hours. This value is optional.

spec.profileCustomizations.thresholdPriorityClassName

Specifies a priority threshold to consider pods for eviction only if their priority is lower than the specified level. Use the **thresholdPriority** field to set a numerical priority threshold (for example, **10000**) or use the **thresholdPriorityClassName** field to specify a certain priority class name (for example, **my-priority-class-name**). If you specify a priority class name, it must already exist or the descheduler will throw an error. Do not set both **thresholdPriority** and **thresholdPriorityClassName**. This value is optional.

spec.evictionLimits.total

Specifies the maximum number of pods to evict during each descheduler run. This value is optional.

spec.profiles

Specifies one or more profiles to enable. Available profiles: **AffinityAndTaints**, **TopologyAndDuplicates**, **LifecycleAndUtilization**, **SoftTopologyAndDuplicates**, **EvictPodsWithLocalStorage**, **EvictPodsWithPVC**, **CompactAndScale**, and **LongLifecycle**. You can enable multiple profiles, but ensure that you do not enable profiles that conflict with each other. The order of the list of profiles is not important.

3. Save the file to apply the changes.

4.9.3.3. Configuring the descheduler interval

You can configure the amount of time between descheduler runs. The default is 3600 seconds (one hour).

Prerequisites

- You are logged in to OpenShift Container Platform as a user with the **cluster-admin** role.

Procedure

1. Edit the **KubeDescheduler** object:

```
$ oc edit kubedeschedulers.operator.openshift.io cluster -n openshift-kube-descheduler-operator
```

2. Update the **deschedulingIntervalSeconds** field to the required value:

```
apiVersion: operator.openshift.io/v1
kind: KubeDescheduler
metadata:
  name: cluster
  namespace: openshift-kube-descheduler-operator
spec:
  deschedulingIntervalSeconds: 3600
...
```

Set the **spec.deschedulingIntervalSeconds** field to the number of seconds you want between descheduler runs. A value of **0** in this field runs the descheduler once and exits.

3. Save the file to apply the changes.

4.9.4. Uninstalling the Kube Descheduler Operator

If you no longer need the Kube Descheduler Operator in your cluster, you can uninstall the Operator and remove its related resources.

4.9.4.1. Uninstalling the descheduler

If you no longer need the descheduler in your cluster, you can remove it by deleting the descheduler instance and uninstalling the Kube Descheduler Operator. You can also delete the **KubeDescheduler** CRD and **openshift-kube-descheduler-operator** namespace.

Prerequisites

- You are logged in to OpenShift Container Platform as a user with the **cluster-admin** role.
- Access to the OpenShift Container Platform web console.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Delete the descheduler instance.
 - a. From the **Ecosystem → Installed Operators** page, click **Kube Descheduler Operator**.
 - b. Select the **Kube Descheduler** tab.

- c. Click the Options menu  next to the **cluster** entry and select **Delete KubeDescheduler**.
 - d. In the confirmation dialog, click **Delete**.
 3. Uninstall the Kube Descheduler Operator.
 - a. Navigate to **Ecosystem → Installed Operators**.
 - b. Click the Options menu  next to the **Kube Descheduler Operator** entry and select **Uninstall Operator**.
 - c. In the confirmation dialog, click **Uninstall**.
 4. Delete the **openshift-kube-descheduler-operator** namespace.
 - a. Navigate to **Administration → Namespaces**.
 - b. Enter **openshift-kube-descheduler-operator** into the filter box.
 - c. Click the Options menu  next to the **openshift-kube-descheduler-operator** entry and select **Delete Namespace**.
 - d. In the confirmation dialog, enter **openshift-kube-descheduler-operator** and click **Delete**.
 5. Delete the **KubeDescheduler** CRD.
 - a. Navigate to **Administration → Custom Resource Definitions**
 - b. Enter **KubeDescheduler** into the filter box.
 - c. Click the Options menu  next to the **KubeDescheduler** entry and select **Delete CustomResourceDefinition**.
 - d. In the confirmation dialog, click **Delete**.

4.10. SECONDARY SCHEDULER

4.10.1. Secondary scheduler overview

You can install the Secondary Scheduler Operator to run a custom secondary scheduler alongside the default scheduler to schedule pods.

4.10.1.1. About the Secondary Scheduler Operator

The Secondary Scheduler Operator for Red Hat OpenShift provides a way to deploy a custom secondary scheduler in OpenShift Container Platform. The secondary scheduler runs alongside the default scheduler to schedule pods. Pod configurations can specify which scheduler to use.

The custom scheduler must have the `/bin/kube-scheduler` binary and be based on the [Kubernetes scheduling framework](#).



IMPORTANT

You can use the Secondary Scheduler Operator to deploy a custom secondary scheduler in OpenShift Container Platform, but Red Hat does not directly support the functionality of the custom secondary scheduler.

The Secondary Scheduler Operator creates the default roles and role bindings required by the secondary scheduler. You can specify which scheduling plugins to enable or disable by configuring the **KubeSchedulerConfiguration** resource for the secondary scheduler.

4.10.2. Secondary Scheduler Operator for Red Hat OpenShift release notes

Review the Secondary Scheduler Operator for Red Hat OpenShift release notes to track its development and learn what is new and changed with each release.

The Secondary Scheduler Operator allows you to deploy a custom secondary scheduler in your OpenShift Container Platform cluster.

For more information, see [About the Secondary Scheduler Operator](#).

4.10.2.1. Release notes for Secondary Scheduler Operator for Red Hat OpenShift 1.5.0

Review the release notes for Secondary Scheduler Operator 1.5.0 to learn what is new and updated with this release.

Issued: 29 October 2025

The following advisory is available for the Secondary Scheduler Operator for Red Hat OpenShift 1.5.0:

- [RHBA-2025:19251](#)

4.10.2.1.1. New features and enhancements

- This release of the Secondary Scheduler Operator updates the Kubernetes version to 1.33.

4.10.2.1.2. Known issues

- Currently, you cannot deploy additional resources, such as config maps, CRDs, or RBAC policies through the Secondary Scheduler Operator. Any resources other than roles and role bindings that are required by your custom secondary scheduler must be applied externally. ([WRKLDS-645](#))

4.10.3. Scheduling pods using a secondary scheduler

You can run a custom secondary scheduler in OpenShift Container Platform by installing the Secondary Scheduler Operator, deploying the secondary scheduler, and setting the secondary scheduler in the pod definition.

4.10.3.1. Installing the Secondary Scheduler Operator

You can install the Secondary Scheduler Operator for Red Hat OpenShift through the OpenShift Container Platform web console to configure a secondary scheduler.

Prerequisites

- You are logged in to OpenShift Container Platform as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Create the required namespace for the Secondary Scheduler Operator for Red Hat OpenShift.
 - a. Navigate to **Administration** → **Namespaces** and click **Create Namespace**.
 - b. Enter **openshift-secondary-scheduler-operator** in the **Name** field and click **Create**.
3. Install the Secondary Scheduler Operator for Red Hat OpenShift.
 - a. Navigate to **Ecosystem** → **Software Catalog**.
 - b. Enter **Secondary Scheduler Operator for Red Hat OpenShift** into the filter box.
 - c. Select the **Secondary Scheduler Operator for Red Hat OpenShift** and click **Install**.
 - d. On the **Install Operator** page:
 - i. The **Update channel** is set to **stable**, which installs the latest stable release of the Secondary Scheduler Operator for Red Hat OpenShift.
 - ii. Select **A specific namespace on the cluster** and select **openshift-secondary-scheduler-operator** from the drop-down menu.
 - iii. Select an **Update approval** strategy.
 - The **Automatic** strategy allows Operator Lifecycle Manager (OLM) to automatically update the Operator when a new version is available.
 - The **Manual** strategy requires a user with appropriate credentials to approve the Operator update.
 - iv. Click **Install**.

Verification

1. Navigate to **Ecosystem** → **Installed Operators**.
2. Verify that **Secondary Scheduler Operator for Red Hat OpenShift** is listed with a **Status** of **Succeeded**.

4.10.3.2. Deploying a secondary scheduler

After you have installed the Secondary Scheduler Operator, you can deploy a secondary scheduler to apply custom placement logic for specific pods.

Prerequisites

- You are logged in to OpenShift Container Platform as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.
- The Secondary Scheduler Operator for Red Hat OpenShift is installed.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Create config map to hold the configuration for the secondary scheduler.
 - a. Navigate to **Workloads → ConfigMaps**.
 - b. Click **Create ConfigMap**.
 - c. In the YAML editor, enter the config map definition that contains the necessary **KubeSchedulerConfiguration** configuration. For example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: "secondary-scheduler-config"
  namespace: "openshift-secondary-scheduler-operator"
data:
  "config.yaml": |
    apiVersion: kubescheduler.config.k8s.io/v1
    kind: KubeSchedulerConfiguration
    leaderElection:
      leaderElect: false
    profiles:
      - schedulerName: secondary-scheduler
    plugins:
      score:
        disabled:
          - name: NodeResourcesBalancedAllocation
          - name: NodeResourcesLeastAllocated
```

where:

metadata.name

Specifies the name of the config map. This is used in the **Scheduler Config** field when creating the **SecondaryScheduler** CR.

metadata.namespace

Specifies the namespace to create the config map in. The namespace must be **openshift-secondary-scheduler-operator**.

data."config.yaml".kind

Specifies the **KubeSchedulerConfiguration** resource for the secondary scheduler. For more information, see [KubeSchedulerConfiguration](#) in the Kubernetes API documentation.

data."config.yaml".profiles.schedulerName

Specifies the name of the secondary scheduler. Pods that set their **spec.schedulerName** field to this value are scheduled with this secondary scheduler.

data."config.yaml".profiles.plugins

Specifies the plugins to enable or disable for the secondary scheduler. For a list default scheduling plugins, see [Scheduling plugins](#) in the Kubernetes documentation.

- d. Click **Create**.
3. Create the **SecondaryScheduler** CR:
 - a. Navigate to **Ecosystem → Installed Operators**.
 - b. Select **Secondary Scheduler Operator for Red Hat OpenShift**
 - c. Select the **Secondary Scheduler** tab and click **Create SecondaryScheduler**.
 - d. The **Name** field defaults to **cluster**; do not change this name.
 - e. The **Scheduler Config** field defaults to **secondary-scheduler-config**. Ensure that this value matches the name of the config map created earlier in this procedure.
 - f. In the **Scheduler Image** field, enter the image name for your custom scheduler.



IMPORTANT

Red Hat does not directly support the functionality of your custom secondary scheduler.

- g. Click **Create**.

4.10.3.3. Scheduling a pod using the secondary scheduler

To schedule a pod by using the secondary scheduler, set the **schedulerName** field in the pod definition.

Prerequisites

- You are logged in to OpenShift Container Platform as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.
- The Secondary Scheduler Operator for Red Hat OpenShift is installed.
- A secondary scheduler is configured.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Workloads → Pods**.
3. Click **Create Pod**.
4. In the YAML editor, enter the desired pod configuration and add the **schedulerName** field:

```
apiVersion: v1
```

```

kind: Pod
metadata:
  name: nginx
  namespace: default
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
  securityContext:
    allowPrivilegeEscalation: false
  capabilities:
    drop: [ALL]
  schedulerName: secondary-scheduler

```

The **spec.schedulerName** field must match the name that is defined in the config map when you configured the secondary scheduler.

5. Click **Create**.

Verification

1. Log in to the OpenShift CLI.
2. Describe the pod using the following command:

```
$ oc describe pod nginx -n default
```

Example output

```

Name:      nginx
Namespace: default
Priority:   0
Node:      ci-ln-t0w4r1k-72292-xkqs4-worker-b-xqkxp/10.0.128.3
...
Events:
  Type     Reason         Age    From          Message
  ---     -
  Normal   Scheduled      12s    secondary-scheduler Successfully assigned default/nginx to
ci-ln-t0w4r1k-72292-xkqs4-worker-b-xqkxp
...

```

3. In the events table, find the event with a message similar to **Successfully assigned <namespace>/<pod_name> to <node_name>**.
4. In the "From" column, verify that the event was generated from the secondary scheduler and not the default scheduler.

**NOTE**

You can also check the **secondary-scheduler-*** pod logs in the **openshift-secondary-scheduler-namespace** to verify that the pod was scheduled by the secondary scheduler.

4.10.4. Uninstalling the Secondary Scheduler Operator

If you no longer need the Secondary Scheduler Operator for Red Hat OpenShift in your cluster, you can uninstall the Operator and remove its related resources.

4.10.4.1. Uninstalling the Secondary Scheduler Operator

You can use the web console to uninstall the Secondary Scheduler Operator for Red Hat OpenShift if you no longer need the Operator in your cluster.

Prerequisites

- You are logged in to OpenShift Container Platform as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.
- The Secondary Scheduler Operator for Red Hat OpenShift is installed.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Uninstall the Secondary Scheduler Operator for Red Hat OpenShift Operator.
 - a. Navigate to **Ecosystem → Installed Operators**.

- 
 b. Click the Options menu next to the **Secondary Scheduler Operator** entry and click **Uninstall Operator**.
- c. In the confirmation dialog, click **Uninstall**.

4.10.4.2. Removing Secondary Scheduler Operator resources

Optionally, remove the custom resource definition (CRD) and associated namespace after the Secondary Scheduler Operator for Red Hat OpenShift is uninstalled. This cleans up all remaining secondary scheduler artifacts.

Prerequisites

- You are logged in to OpenShift Container Platform as a user with the **cluster-admin** role.
- You have access to the OpenShift Container Platform web console.

Procedure

1. Log in to the OpenShift Container Platform web console.

2. Remove the CRD that was installed by the Secondary Scheduler Operator:

a. Navigate to **Administration** → **CustomResourceDefinitions**.

b. Enter **SecondaryScheduler** in the **Name** field to filter the CRDs.

c. Click the Options menu  next to the **SecondaryScheduler** CRD and select **Delete Custom Resource Definition**:

3. Remove the **openshift-secondary-scheduler-operator** namespace.

a. Navigate to **Administration** → **Namespaces**.

b. Click the Options menu  next to the **openshift-secondary-scheduler-operator** and select **Delete Namespace**.

c. In the confirmation dialog, enter **openshift-secondary-scheduler-operator** in the field and click **Delete**.

CHAPTER 5. USING JOBS AND DAEMON SETS

5.1. RUNNING BACKGROUND TASKS ON NODES AUTOMATICALLY WITH DAEMON SETS

As an administrator, you can create and use daemon sets to run replicas of a pod on specific or all nodes in an OpenShift Container Platform cluster.

A daemon set ensures that all (or some) nodes run a copy of a pod. As nodes are added to the cluster, pods are added to the cluster. As nodes are removed from the cluster, those pods are removed through garbage collection. Deleting a daemon set will clean up the pods it created.

You can use daemon sets to create shared storage, run a logging pod on every node in your cluster, or deploy a monitoring agent on every node.

For security reasons, the cluster administrators and the project administrators can create daemon sets.

For more information on daemon sets, see the [Kubernetes documentation](#).



IMPORTANT

Daemon set scheduling is incompatible with project's default node selector. If you fail to disable it, the daemon set gets restricted by merging with the default node selector. This results in frequent pod recreates on the nodes that got unselected by the merged node selector, which in turn puts unwanted load on the cluster.

5.1.1. Scheduled by default scheduler

A daemon set ensures that all eligible nodes run a copy of a pod. Normally, the node that a pod runs on is selected by the Kubernetes scheduler. However, daemon set pods are created and scheduled by the daemon set controller. That introduces the following issues:

- Inconsistent pod behavior: Normal pods waiting to be scheduled are created and in Pending state, but daemon set pods are not created in **Pending** state. This is confusing to the user.
- Pod preemption is handled by default scheduler. When preemption is enabled, the daemon set controller will make scheduling decisions without considering pod priority and preemption.

The **ScheduleDaemonSetPods** feature, enabled by default in OpenShift Container Platform, lets you schedule daemon sets using the default scheduler instead of the daemon set controller, by adding the **NodeAffinity** term to the daemon set pods, instead of the **spec.nodeName** term. The default scheduler is then used to bind the pod to the target host. If node affinity of the daemon set pod already exists, it is replaced. The daemon set controller only performs these operations when creating or modifying daemon set pods, and no changes are made to the **spec.template** of the daemon set.

```
kind: Pod
apiVersion: v1
metadata:
  name: hello-node-6fbccf8d9-9tmzr
#...
spec:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
```

```
- matchFields:
- key: metadata.name
  operator: In
  values:
- target-host-name
```

```
#...
```

In addition, a **node.kubernetes.io/unschedulable:NoSchedule** toleration is added automatically to daemon set pods. The default scheduler ignores unschedulable Nodes when scheduling daemon set pods.

5.1.2. Creating daemonsets

When creating daemon sets, the **nodeSelector** field is used to indicate the nodes on which the daemon set should deploy replicas.

Prerequisites

- Before you start using daemon sets, disable the default project-wide node selector in your namespace, by setting the namespace annotation **openshift.io/node-selector** to an empty string:

```
$ oc patch namespace myproject -p \
  '{"metadata": {"annotations": {"openshift.io/node-selector": ""}}}'
```

TIP

You can alternatively apply the following YAML to disable the default project-wide node selector for a namespace:

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
  annotations:
    openshift.io/node-selector: ""
#...
```

- If you are creating a new project, overwrite the default node selector:

```
$ oc adm new-project <name> --node-selector=""
```

Procedure

To create a daemon set:

1. Define the daemon set yaml file:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: hello-daemonset
spec:
```

```

selector:
  matchLabels:
    name: hello-daemonset ❶
template:
  metadata:
    labels:
      name: hello-daemonset ❷
  spec:
    nodeSelector: ❸
      role: worker
    containers:
      - image: openshift/hello-openshift
        imagePullPolicy: Always
        name: registry
        ports:
          - containerPort: 80
            protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
    serviceAccount: default
    terminationGracePeriodSeconds: 10
#...
```

- ❶ The label selector that determines which pods belong to the daemon set.
- ❷ The pod template's label selector. Must match the label selector above.
- ❸ The node selector that determines on which nodes pod replicas should be deployed. A matching label must be present on the node.

2. Create the daemon set object:

```
$ oc create -f daemonset.yaml
```

3. To verify that the pods were created, and that each node has a pod replica:

a. Find the daemonset pods:

```
$ oc get pods
```

Example output

```

hello-daemonset-cx6md 1/1   Running 0    2m
hello-daemonset-e3md9 1/1   Running 0    2m
```

b. View the pods to verify the pod has been placed onto the node:

```
$ oc describe pod/hello-daemonset-cx6md | grep Node
```

Example output

```
Node:      openshift-node01.hostname.com/10.14.20.134
```

```
$ oc describe pod/hello-daemonset-e3md9|grep Node
```

Example output

```
Node:      openshift-node02.hostname.com/10.14.20.137
```

IMPORTANT

- If you update a daemon set pod template, the existing pod replicas are not affected.
- If you delete a daemon set and then create a new daemon set with a different template but the same label selector, it recognizes any existing pod replicas as having matching labels and thus does not update them or create new replicas despite a mismatch in the pod template.
- If you change node labels, the daemon set adds pods to nodes that match the new labels and deletes pods from nodes that do not match the new labels.

To update a daemon set, force new pod replicas to be created by deleting the old replicas or nodes.

5.2. RUNNING TASKS IN PODS USING JOBS

A *job* executes a task in your OpenShift Container Platform cluster.

A job tracks the overall progress of a task and updates its status with information about active, succeeded, and failed pods. Deleting a job will clean up any pod replicas it created. Jobs are part of the Kubernetes API, which can be managed with **oc** commands like other object types.

Sample Job specification

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  parallelism: 1 1
  completions: 1 2
  activeDeadlineSeconds: 1800 3
  backoffLimit: 6 4
  template: 5
    metadata:
      name: pi
    spec:
      containers:
        - name: pi
          image: perl
          command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: OnFailure 6
#...
```

- 1** The pod replicas a job should run in parallel.

- 2 Successful pod completions are needed to mark a job completed.
- 3 The maximum duration the job can run.
- 4 The number of retries for a job.
- 5 The template for the pod the controller creates.
- 6 The restart policy of the pod.

Additional resources

- [Jobs \(Kubernetes documentation\)](#)

5.2.1. Understanding jobs and cron jobs

A job tracks the overall progress of a task and updates its status with information about active, succeeded, and failed pods. Deleting a job cleans up any pods it created. Jobs are part of the Kubernetes API, which can be managed with **oc** commands like other object types.

There are two possible resource types that allow creating run-once objects in OpenShift Container Platform:

Job

A regular job is a run-once object that creates a task and ensures the job finishes.

There are three main types of task suitable to run as a job:

- Non-parallel jobs:
 - A job that starts only one pod, unless the pod fails.
 - The job is complete as soon as its pod terminates successfully.
- Parallel jobs with a fixed completion count:
 - a job that starts multiple pods.
 - The job represents the overall task and is complete when there is one successful pod for each value in the range **1** to the **completions** value.
- Parallel jobs with a work queue:
 - A job with multiple parallel worker processes in a given pod.
 - OpenShift Container Platform coordinates pods to determine what each should work on or use an external queue service.
 - Each pod is independently capable of determining whether or not all peer pods are complete and that the entire job is done.
 - When any pod from the job terminates with success, no new pods are created.
 - When at least one pod has terminated with success and all pods are terminated, the job is successfully completed.

- When any pod has exited with success, no other pod should be doing any work for this task or writing any output. Pods should all be in the process of exiting.
For more information about how to make use of the different types of job, see [Job Patterns](#) in the Kubernetes documentation.

Cron job

A job can be scheduled to run multiple times, using a cron job.

A *cron job* builds on a regular job by allowing you to specify how the job should be run. Cron jobs are part of the [Kubernetes](#) API, which can be managed with **oc** commands like other object types.

Cron jobs are useful for creating periodic and recurring tasks, like running backups or sending emails. Cron jobs can also schedule individual tasks for a specific time, such as if you want to schedule a job for a low activity period. A cron job creates a **Job** object based on the timezone configured on the control plane node that runs the cronjob controller.



WARNING

A cron job creates a **Job** object approximately once per execution time of its schedule, but there are circumstances in which it fails to create a job or two jobs might be created. Therefore, jobs must be idempotent and you must configure history limits.

5.2.1.1. Understanding how to create jobs

Both resource types require a job configuration that consists of the following key parts:

- A pod template, which describes the pod that OpenShift Container Platform creates.
- The **parallelism** parameter, which specifies how many pods running in parallel at any point in time should execute a job.
 - For non-parallel jobs, leave unset. When unset, defaults to **1**.
- The **completions** parameter, specifying how many successful pod completions are needed to finish a job.
 - For non-parallel jobs, leave unset. When unset, defaults to **1**.
 - For parallel jobs with a fixed completion count, specify a value.
 - For parallel jobs with a work queue, leave unset. When unset defaults to the **parallelism** value.

5.2.1.2. Understanding how to set a maximum duration for jobs

When defining a job, you can define its maximum duration by setting the **activeDeadlineSeconds** field. It is specified in seconds and is not set by default. When not set, there is no maximum duration enforced.

The maximum duration is counted from the time when a first pod gets scheduled in the system, and defines how long a job can be active. It tracks overall time of an execution. After reaching the specified timeout, the job is terminated by OpenShift Container Platform.

5.2.1.3. Understanding how to set a job back off policy for pod failure

A job can be considered failed, after a set amount of retries due to a logical error in configuration or other similar reasons. Failed pods associated with the job are recreated by the controller with an exponential back off delay (**10s, 20s, 40s** ...) capped at six minutes. The limit is reset if no new failed pods appear between controller checks.

Use the **spec.backoffLimit** parameter to set the number of retries for a job.

5.2.1.4. Understanding how to configure a cron job to remove artifacts

Cron jobs can leave behind artifact resources such as jobs or pods. As a user it is important to configure history limits so that old jobs and their pods are properly cleaned. There are two fields within cron job's spec responsible for that:

- **.spec.successfulJobsHistoryLimit**. The number of successful finished jobs to retain (defaults to 3).
- **.spec.failedJobsHistoryLimit**. The number of failed finished jobs to retain (defaults to 1).

TIP

- Delete cron jobs that you no longer need:

```
$ oc delete cronjob/<cron_job_name>
```

Doing this prevents them from generating unnecessary artifacts.

- You can suspend further executions by setting the **spec.suspend** to true. All subsequent executions are suspended until you reset to **false**.

5.2.1.5. Known limitations

The job specification restart policy only applies to the *pods*, and not the *job controller*. However, the job controller is hard-coded to keep retrying jobs to completion.

As such, **restartPolicy: Never** or **--restart=Never** results in the same behavior as **restartPolicy: OnFailure** or **--restart=OnFailure**. That is, when a job fails it is restarted automatically until it succeeds (or is manually discarded). The policy only sets which subsystem performs the restart.

With the **Never** policy, the *job controller* performs the restart. With each attempt, the job controller increments the number of failures in the job status and create new pods. This means that with each failed attempt, the number of pods increases.

With the **OnFailure** policy, *kubelet* performs the restart. Each attempt does not increment the number of failures in the job status. In addition, kubelet will retry failed jobs starting pods on the same nodes.

5.2.2. Creating jobs

You create a job in OpenShift Container Platform by creating a job object.

Procedure

To create a job:

1. Create a YAML file similar to the following:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  parallelism: 1 1
  completions: 1 2
  activeDeadlineSeconds: 1800 3
  backoffLimit: 6 4
  template: 5
    metadata:
      name: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: OnFailure 6
#...
```

- 1** Optional: Specify how many pod replicas a job should run in parallel; defaults to **1**.
 - For non-parallel jobs, leave unset. When unset, defaults to **1**.
- 2** Optional: Specify how many successful pod completions are needed to mark a job completed.
 - For non-parallel jobs, leave unset. When unset, defaults to **1**.
 - For parallel jobs with a fixed completion count, specify the number of completions.
 - For parallel jobs with a work queue, leave unset. When unset defaults to the **parallelism** value.
- 3** Optional: Specify the maximum duration the job can run.
- 4** Optional: Specify the number of retries for a job. This field defaults to six.
- 5** Specify the template for the pod the controller creates.
- 6** Specify the restart policy of the pod:
 - **Never**. Do not restart the job.
 - **OnFailure**. Restart the job only if it fails.
 - **Always**. Always restart the job.

For details on how OpenShift Container Platform uses restart policy with failed containers, see the [Example States](#) in the Kubernetes documentation.

2. Create the job:

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

**NOTE**

You can also create and launch a job from a single command using **oc create job**. The following command creates and launches a job similar to the one specified in the previous example:

```
$ oc create job pi --image=perl -- perl -Mbignum=bpi -wle 'print bpi(2000)'
```

5.2.3. Creating cron jobs

You create a cron job in OpenShift Container Platform by creating a job object.

Procedure

To create a cron job:

1. Create a YAML file similar to the following:

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: pi
spec:
  schedule: "*/1 * * * *" 1
  timeZone: Etc/UTC 2
  concurrencyPolicy: "Replace" 3
  startingDeadlineSeconds: 200 4
  suspend: true 5
  successfulJobsHistoryLimit: 3 6
  failedJobsHistoryLimit: 1 7
  jobTemplate: 8
    spec:
      template:
        metadata:
          labels: 9
            parent: "cronjobpi"
        spec:
          containers:
            - name: pi
              image: perl
              command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
          restartPolicy: OnFailure 10
#...
```

- 1 Schedule for the job specified in [cron format](#). In this example, the job will run every minute.
- 2 An optional time zone for the schedule. See [List of tz database time zones](#) for valid options. If not specified, the Kubernetes controller manager interprets the schedule relative to its local time zone.

- 3 An optional concurrency policy, specifying how to treat concurrent jobs within a cron job. Only one of the following concurrent policies may be specified. If not specified, this
 - **Allow** allows cron jobs to run concurrently.
 - **Forbid** forbids concurrent runs, skipping the next run if the previous has not finished yet.
 - **Replace** cancels the currently running job and replaces it with a new one.
- 4 An optional deadline (in seconds) for starting the job if it misses its scheduled time for any reason. Missed jobs executions will be counted as failed ones. If not specified, there is no deadline.
- 5 An optional flag allowing the suspension of a cron job. If set to **true**, all subsequent executions will be suspended.
- 6 The number of successful finished jobs to retain (defaults to 3).
- 7 The number of failed finished jobs to retain (defaults to 1).
- 8 Job template. This is similar to the job example.
- 9 Sets a label for jobs spawned by this cron job.
- 10 The restart policy of the pod. This does not apply to the job controller.

2. Create the cron job:

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

NOTE

You can also create and launch a cron job from a single command using **oc create cronjob**. The following command creates and launches a cron job similar to the one specified in the previous example:

```
$ oc create cronjob pi --image=perl --schedule='*/1 * * * *' -- perl -Mbignum=bpi -wle 'print bpi(2000)'
```

With **oc create cronjob**, the **--schedule** option accepts schedules in [cron format](#).

CHAPTER 6. WORKING WITH NODES

6.1. VIEWING AND LISTING THE NODES IN YOUR OPENSIFT CONTAINER PLATFORM CLUSTER

You can list all the nodes in your cluster to obtain information such as status, age, memory usage, and details about the nodes.

When you perform node management operations, the CLI interacts with node objects that are representations of actual node hosts. The master uses the information from node objects to validate nodes with health checks.

6.1.1. About listing all the nodes in a cluster

You can get detailed information on the nodes in the cluster.

- The following command lists all nodes:

```
$ oc get nodes
```

The following example is a cluster with healthy nodes:

```
$ oc get nodes
```

Example output

NAME	STATUS	ROLES	AGE	VERSION
master.example.com	Ready	master	7h	v1.33.4
node1.example.com	Ready	worker	7h	v1.33.4
node2.example.com	Ready	worker	7h	v1.33.4

The following example is a cluster with one unhealthy node:

```
$ oc get nodes
```

Example output

NAME	STATUS	ROLES	AGE	VERSION
master.example.com	Ready	master	7h	v1.33.4
node1.example.com	NotReady,SchedulingDisabled	worker	7h	v1.33.4
node2.example.com	Ready	worker	7h	v1.33.4

The conditions that trigger a **NotReady** status are shown later in this section.

- The **-o wide** option provides additional information on nodes.

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

Example output

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP

OS-IMAGE RUNTIME	KERNEL-VERSION	CONTAINER-
master.example.com Ready master 171m v1.33.4 10.0.129.108 <none> Red Hat Enterprise Linux CoreOS 48.83.202103210901-0 (Ootpa) 4.20.0-240.15.1.el8_3.x86_64 cri-o://1.33.4-30.rhaos4.10.gitf2f339d.el8-dev		
node1.example.com Ready worker 72m v1.33.4 10.0.129.222 <none> Red Hat Enterprise Linux CoreOS 48.83.202103210901-0 (Ootpa) 4.20.0-240.15.1.el8_3.x86_64 cri-o://1.33.4-30.rhaos4.10.gitf2f339d.el8-dev		
node2.example.com Ready worker 164m v1.33.4 10.0.142.150 <none> Red Hat Enterprise Linux CoreOS 48.83.202103210901-0 (Ootpa) 4.20.0-240.15.1.el8_3.x86_64 cri-o://1.33.4-30.rhaos4.10.gitf2f339d.el8-dev		

- The following command lists information about a single node:

```
$ oc get node <node>
```

For example:

```
$ oc get node node1.example.com
```

Example output

NAME	STATUS	ROLES	AGE	VERSION
node1.example.com	Ready	worker	7h	v1.33.4

- The following command provides more detailed information about a specific node, including the reason for the current condition:

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

For example:

```
$ oc describe node node1.example.com
```



NOTE

The following example contains some values that are specific to OpenShift Container Platform on AWS.

Example output

```
Name:      node1.example.com 1
Roles:     worker 2
Labels:    kubernetes.io/os=linux
           kubernetes.io/hostname=ip-10-0-131-14
           kubernetes.io/arch=amd64 3
           node-role.kubernetes.io/worker=
           node.kubernetes.io/instance-type=m4.large
           node.openshift.io/os_id=rhcos
           node.openshift.io/os_version=4.5
           region=east
           topology.kubernetes.io/region=us-east-1
```

```

    topology.kubernetes.io/zone=us-east-1a
Annotations:    cluster.k8s.io/machine: openshift-machine-api/ahardin-worker-us-east-2a-
q5dzc 4
    machineconfiguration.openshift.io/currentConfig: worker-
309c228e8b3a92e2235edd544c62fea8
    machineconfiguration.openshift.io/desiredConfig: worker-
309c228e8b3a92e2235edd544c62fea8
    machineconfiguration.openshift.io/state: Done
    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp: Wed, 13 Feb 2019 11:05:57 -0500
Taints:        <none> 5
Unschedulable: false
Conditions:    6
  Type           Status  LastHeartbeatTime           LastTransitionTime          Reason
  Message
  ----
  OutOfDisk      False   Wed, 13 Feb 2019 15:09:42 -0500   Wed, 13 Feb 2019 11:05:57 -
0500   KubeletHasSufficientDisk   kubelet has sufficient disk space available
  MemoryPressure False   Wed, 13 Feb 2019 15:09:42 -0500   Wed, 13 Feb 2019 11:05:57
-0500   KubeletHasSufficientMemory kubelet has sufficient memory available
  DiskPressure   False   Wed, 13 Feb 2019 15:09:42 -0500   Wed, 13 Feb 2019 11:05:57 -
0500   KubeletHasNoDiskPressure  kubelet has no disk pressure
  PIDPressure    False   Wed, 13 Feb 2019 15:09:42 -0500   Wed, 13 Feb 2019 11:05:57 -
0500   KubeletHasSufficientPID   kubelet has sufficient PID available
  Ready          True    Wed, 13 Feb 2019 15:09:42 -0500   Wed, 13 Feb 2019 11:07:09 -0500
  KubeletReady   kubelet is posting ready status
Addresses:    7
  InternalIP:  10.0.140.16
  InternalDNS:  ip-10-0-140-16.us-east-2.compute.internal
  Hostname:     ip-10-0-140-16.us-east-2.compute.internal
Capacity:    8
  attachable-volumes-aws-ebs: 39
  cpu:                      2
  hugepages-1Gi:            0
  hugepages-2Mi:            0
  memory:                    8172516Ki
  pods:                      250
Allocatable:
  attachable-volumes-aws-ebs: 39
  cpu:                      1500m
  hugepages-1Gi:            0
  hugepages-2Mi:            0
  memory:                    7558116Ki
  pods:                      250
System Info:  9
  Machine ID:      63787c9534c24fde9a0cde35c13f1f66
  System UUID:     EC22BF97-A006-4A58-6AF8-0A38DEEA122A
  Boot ID:         f24ad37d-2594-46b4-8830-7f7555918325
  Kernel Version:   3.10.0-957.5.1.el7.x86_64
  OS Image:         Red Hat Enterprise Linux CoreOS 410.8.20190520.0 (Ootpa)
  Operating System: linux
  Architecture:     amd64
  Container Runtime Version: cri-o://1.33.4-0.6.dev.rhaos4.3.git9ad059b.el8-rc2
  Kubelet Version:   v1.33.4
  Kube-Proxy Version: v1.33.4
  PodCIDR:           10.128.4.0/24

```

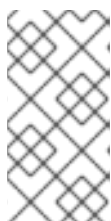
```

ProviderID:                aws:///us-east-2a/i-04e87b31dc6b3e171
Non-terminated Pods:      (12 in total) 10
  Namespace                Name                CPU Requests  CPU Limits
  Memory Requests  Memory Limits
  -----
  -----
  openshift-cluster-node-tuning-operator tuned-hdl5q          0 (0%)      0 (0%)      0
  (0%)      0 (0%)
  openshift-dns                dns-default-l69zr          0 (0%)      0 (0%)      0 (0%)
  0 (0%)
  openshift-image-registry      node-ca-9hmcg          0 (0%)      0 (0%)      0
  (0%)      0 (0%)
  openshift-ingress            router-default-76455c45c-c5ptv  0 (0%)      0 (0%)      0
  (0%)      0 (0%)
  openshift-machine-config-operator machine-config-daemon-cvqw9      20m (1%)      0
  (0%)  50Mi (0%)  0 (0%)
  openshift-marketplace          community-operators-f67fh          0 (0%)      0 (0%)
  0 (0%)      0 (0%)
  openshift-monitoring          alertmanager-main-0          50m (3%)      50m (3%)
  210Mi (2%)  10Mi (0%)
  openshift-monitoring          node-exporter-l7q8d          10m (0%)      20m (1%)
  20Mi (0%)  40Mi (0%)
  openshift-monitoring          prometheus-adapter-75d769c874-hvb85  0 (0%)      0
  (0%)  0 (0%)  0 (0%)
  openshift-multus              multus-kw8w5          0 (0%)      0 (0%)      0 (0%)
  0 (0%)
  openshift-ovn-kubernetes      ovnkube-node-t4dsn          80m (0%)
  0 (0%)  1630Mi (0%)  0 (0%)
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource           Requests  Limits
-----
cpu                380m (25%)  270m (18%)
memory             880Mi (11%)  250Mi (3%)
attachable-volumes-aws-ebs  0          0
Events: 11
Type    Reason                Age          From          Message
----
Normal  NodeHasSufficientPID  6d (x5 over 6d)  kubelet, m01.example.com Node
m01.example.com status is now: NodeHasSufficientPID
Normal  NodeAllocatableEnforced  6d          kubelet, m01.example.com Updated Node
Allocatable limit across pods
Normal  NodeHasSufficientMemory  6d (x6 over 6d)  kubelet, m01.example.com Node
m01.example.com status is now: NodeHasSufficientMemory
Normal  NodeHasNoDiskPressure  6d (x6 over 6d)  kubelet, m01.example.com Node
m01.example.com status is now: NodeHasNoDiskPressure
Normal  NodeHasSufficientDisk  6d (x6 over 6d)  kubelet, m01.example.com Node
m01.example.com status is now: NodeHasSufficientDisk
Normal  NodeHasSufficientPID  6d          kubelet, m01.example.com Node
m01.example.com status is now: NodeHasSufficientPID
Normal  Starting              6d          kubelet, m01.example.com Starting kubelet.
#...

```

1 The name of the node.

- 2 The role of the node, either **master** or **worker**.
- 3 The labels applied to the node.
- 4 The annotations applied to the node.
- 5 The taints applied to the node.
- 6 The node conditions and status. The **conditions** stanza lists the **Ready**, **PIDPressure**, **MemoryPressure**, **DiskPressure** and **OutOfDisk** status. These condition are described later in this section.
- 7 The IP address and hostname of the node.
- 8 The pod resources and allocatable resources.
- 9 Information about the node host.
- 10 The pods on the node.
- 11 The events reported by the node.



NOTE

The control plane label is not automatically added to newly created or updated master nodes. If you want to use the control plane label for your nodes, you can manually configure the label. For more information, see *Understanding how to update labels on nodes* in the *Additional resources* section.

Among the information shown for nodes, the following node conditions appear in the output of the commands shown in this section:

Table 6.1. Node Conditions

Condition	Description
Ready	If true , the node is healthy and ready to accept pods. If false , the node is not healthy and is not accepting pods. If unknown , the node controller has not received a heartbeat from the node for the node-monitor-grace-period (the default is 40 seconds).
DiskPressure	If true , the disk capacity is low.
MemoryPressure	If true , the node memory is low.
PIDPressure	If true , there are too many processes on the node.
OutOfDisk	If true , the node has insufficient free space on the node for adding new pods.
NetworkUnavailable	If true , the network for the node is not correctly configured.

Condition	Description
NotReady	If true , one of the underlying components, such as the container runtime or network, is experiencing issues or is not yet configured.
SchedulingDisabled	Pods cannot be scheduled for placement on the node.

Additional resources

- [Understanding how to update labels on nodes](#)

6.1.2. Listing pods on a node in your cluster

You can list all the pods on a specific node.

Procedure

- To list all or selected pods on selected nodes:

```
$ oc get pod --selector=<nodeSelector>
```

```
$ oc get pod --selector=kubernetes.io/os
```

Or:

```
$ oc get pod -l=<nodeSelector>
```

```
$ oc get pod -l kubernetes.io/os=linux
```

- To list all pods on a specific node, including terminated pods:

```
$ oc get pod --all-namespaces --field-selector=spec.nodeName=<nodename>
```

6.1.3. Viewing memory and CPU usage statistics on your nodes

You can display usage statistics about nodes, which provide the runtime environments for containers. These usage statistics include CPU, memory, and storage consumption.

Prerequisites

- You must have **cluster-reader** permission to view the usage statistics.
- Metrics must be installed to view the usage statistics.

Procedure

- To view the usage statistics:

```
$ oc adm top nodes
```

Example output

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
ip-10-0-12-143.ec2.compute.internal	1503m	100%	4533Mi	61%
ip-10-0-132-16.ec2.compute.internal	76m	5%	1391Mi	18%
ip-10-0-140-137.ec2.compute.internal	398m	26%	2473Mi	33%
ip-10-0-142-44.ec2.compute.internal	656m	43%	6119Mi	82%
ip-10-0-146-165.ec2.compute.internal	188m	12%	3367Mi	45%
ip-10-0-19-62.ec2.compute.internal	896m	59%	5754Mi	77%
ip-10-0-44-193.ec2.compute.internal	632m	42%	5349Mi	72%

- To view the usage statistics for nodes with labels:

```
$ oc adm top node --selector=
```

You must choose the selector (label query) to filter on. Supports `=`, `==`, and `!=`.

6.2. WORKING WITH NODES

Additional resources

As an administrator, you can perform several tasks to make your clusters more efficient.

6.2.1. Understanding how to evacuate pods on nodes

Evacuating pods allows you to migrate all or selected pods from a given node or nodes.

You can only evacuate pods backed by a replication controller. The replication controller creates new pods on other nodes and removes the existing pods from the specified node(s).

Bare pods, meaning those not backed by a replication controller, are unaffected by default. You can evacuate a subset of pods by specifying a pod-selector. Pod selectors are based on labels, so all the pods with the specified label will be evacuated.

Procedure

1. Mark the nodes unschedulable before performing the pod evacuation.

- a. Mark the node as unschedulable:

```
$ oc adm cordon <node1>
```

Example output

```
node/<node1> cordoned
```

- b. Check that the node status is **Ready,SchedulingDisabled**:

```
$ oc get node <node1>
```

Example output

NAME	STATUS	ROLES	AGE	VERSION
<node1>	Ready,SchedulingDisabled	worker	1d	v1.33.4

2. Evacuate the pods using one of the following methods:

- Evacuate all or selected pods on one or more nodes:

```
$ oc adm drain <node1> <node2> [--pod-selector=<pod_selector>]
```

- Force the deletion of bare pods using the **--force** option. When set to **true**, deletion continues even if there are pods not managed by a replication controller, replica set, job, daemon set, or stateful set:

```
$ oc adm drain <node1> <node2> --force=true
```

- Set a period of time in seconds for each pod to terminate gracefully, use **--grace-period**. If negative, the default value specified in the pod will be used:

```
$ oc adm drain <node1> <node2> --grace-period=-1
```

- Ignore pods managed by daemon sets using the **--ignore-daemonsets** flag set to **true**:

```
$ oc adm drain <node1> <node2> --ignore-daemonsets=true
```

- Set the length of time to wait before giving up using the **--timeout** flag. A value of **0** sets an infinite length of time:

```
$ oc adm drain <node1> <node2> --timeout=5s
```

- Delete pods even if there are pods using **emptyDir** volumes by setting the **--delete-emptydir-data** flag to **true**. Local data is deleted when the node is drained:

```
$ oc adm drain <node1> <node2> --delete-emptydir-data=true
```

- List objects that will be migrated without actually performing the evacuation, using the **--dry-run** option set to **true**:

```
$ oc adm drain <node1> <node2> --dry-run=true
```

Instead of specifying specific node names (for example, **<node1> <node2>**), you can use the **--selector=<node_selector>** option to evacuate pods on selected nodes.

3. Mark the node as schedulable when done.

```
$ oc adm uncordon <node1>
```

6.2.2. Understanding how to update labels on nodes

You can update any label on a node.

Node labels are not persisted after a node is deleted even if the node is backed up by a Machine.



NOTE

Any change to a **MachineSet** object is not applied to existing machines owned by the compute machine set. For example, labels edited or added to an existing **MachineSet** object are not propagated to existing machines and nodes associated with the compute machine set.

- The following command adds or updates labels on a node:

```
$ oc label node <node> <key_1>=<value_1> ... <key_n>=<value_n>
```

For example:

```
$ oc label nodes webconsole-7f7f6 unhealthy=true
```

TIP

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

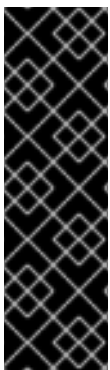
```
kind: Node
apiVersion: v1
metadata:
  name: webconsole-7f7f6
  labels:
    unhealthy: 'true'
#...
```

- The following command updates all pods in the namespace:

```
$ oc label pods --all <key_1>=<value_1>
```

For example:

```
$ oc label pods --all status=unhealthy
```



IMPORTANT

In OpenShift Container Platform 4.12 and later, newly installed clusters include both the **node-role.kubernetes.io/control-plane** and **node-role.kubernetes.io/master** labels on control plane nodes by default.

In OpenShift Container Platform versions earlier than 4.12, the **node-role.kubernetes.io/control-plane** label is not added by default. Therefore, you must manually add the **node-role.kubernetes.io/control-plane** label to control plane nodes in clusters upgraded from earlier versions.

6.2.3. Understanding how to mark nodes as unschedulable or schedulable

By default, healthy nodes with a **Ready** status are marked as schedulable, which means that you can place new pods on the node. Manually marking a node as unschedulable blocks any new pods from being scheduled on the node. Existing pods on the node are not affected.

- The following command marks a node or nodes as unschedulable:

Example output

```
$ oc adm cordon <node>
```

For example:

```
$ oc adm cordon node1.example.com
```

Example output

```
node/node1.example.com cordoned
```

NAME	LABELS	STATUS
node1.example.com	kubernetes.io/hostname=node1.example.com	Ready,SchedulingDisabled

- The following command marks a currently unschedulable node or nodes as schedulable:

```
$ oc adm uncordon <node1>
```

Alternatively, instead of specifying specific node names (for example, **<node>**), you can use the **--selector=<node_selector>** option to mark selected nodes as schedulable or unschedulable.

6.2.4. Handling errors in single-node OpenShift clusters when the node reboots without draining application pods

In single-node OpenShift clusters and in OpenShift Container Platform clusters in general, a situation can arise where a node reboot occurs without first draining the node. This can occur where an application pod requesting devices fails with the **UnexpectedAdmissionError** error. **Deployment**, **ReplicaSet**, or **DaemonSet** errors are reported because the application pods that require those devices start before the pod serving those devices. You cannot control the order of pod restarts.

While this behavior is to be expected, it can cause a pod to remain on the cluster even though it has failed to deploy successfully. The pod continues to report **UnexpectedAdmissionError**. This issue is mitigated by the fact that application pods are typically included in a **Deployment**, **ReplicaSet**, or **DaemonSet**. If a pod is in this error state, it is of little concern because another instance should be running. Belonging to a **Deployment**, **ReplicaSet**, or **DaemonSet** guarantees the successful creation and execution of subsequent pods and ensures the successful deployment of the application.

There is ongoing work upstream to ensure that such pods are gracefully terminated. Until that work is resolved, run the following command for a single-node OpenShift cluster to remove the failed pods:

```
$ oc delete pods --field-selector status.phase=Failed -n <POD_NAMESPACE>
```



NOTE

The option to drain the node is unavailable for single-node OpenShift clusters.

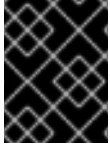
Additional resources

- [Understanding how to evacuate pods on nodes](#)

6.2.5. Deleting nodes

6.2.5.1. Deleting nodes from a cluster

To delete a node from the OpenShift Container Platform cluster, scale down the appropriate **MachineSet** object.



IMPORTANT

When a cluster is integrated with a cloud provider, you must delete the corresponding machine to delete a node. Do not try to use the **oc delete node** command for this task.

When you delete a node by using the CLI, the node object is deleted in Kubernetes, but the pods that exist on the node are not deleted. Any bare pods that are not backed by a replication controller become inaccessible to OpenShift Container Platform. Pods backed by replication controllers are rescheduled to other available nodes. You must delete local manifest pods.



NOTE

If you are running cluster on bare metal, you cannot delete a node by editing **MachineSet** objects. Compute machine sets are only available when a cluster is integrated with a cloud provider. Instead you must unschedule and drain the node before manually deleting it.

Procedure

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

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

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

2. Scale down the compute machine set by using one of the following methods:

- Specify the number of replicas to scale down to by running the following command:

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

- Edit the compute machine set custom resource by running the following command:

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

Example output

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  # ...
  name: <machine-set-name>
  namespace: openshift-machine-api
  # ...
spec:
```

```
replicas: 2 1
# ...
```

- 1** Specify the number of replicas to scale down to.

Additional resources

- [Manually scaling a compute machine set](#)

6.2.5.2. Deleting nodes from a bare metal cluster

When you delete a node using the CLI, the node object is deleted in Kubernetes, but the pods that exist on the node are not deleted. Any bare pods not backed by a replication controller become inaccessible to OpenShift Container Platform. Pods backed by replication controllers are rescheduled to other available nodes. You must delete local manifest pods.

Procedure

Delete a node from an OpenShift Container Platform cluster running on bare metal by completing the following steps:

1. Mark the node as unschedulable:

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

2. Drain all pods on the node:

```
$ oc adm drain <node_name> --force=true
```

This step might fail if the node is offline or unresponsive. Even if the node does not respond, it might still be running a workload that writes to shared storage. To avoid data corruption, power down the physical hardware before you proceed.

3. Delete the node from the cluster:

```
$ oc delete node <node_name>
```

Although the node object is now deleted from the cluster, it can still rejoin the cluster after reboot or if the kubelet service is restarted. To permanently delete the node and all its data, you must [decommission the node](#).

4. If you powered down the physical hardware, turn it back on so that the node can rejoin the cluster.

6.3. MANAGING NODES

OpenShift Container Platform uses a KubeletConfig custom resource (CR) to manage the configuration of nodes. By creating an instance of a **KubeletConfig** object, a managed machine config is created to override setting on the node.

**NOTE**

Logging in to remote machines for the purpose of changing their configuration is not supported.

6.3.1. Modifying nodes

To make configuration changes to a cluster, or machine pool, you must create a custom resource definition (CRD), or **kubeletConfig** object. OpenShift Container Platform uses the Machine Config Controller to watch for changes introduced through the CRD to apply the changes to the cluster.

**NOTE**

Because the fields in a **kubeletConfig** object are passed directly to the kubelet from upstream Kubernetes, the validation of those fields is handled directly by the kubelet itself. Please refer to the relevant Kubernetes documentation for the valid values for these fields. Invalid values in the **kubeletConfig** object can render cluster nodes unusable.

Procedure

1. Obtain the label associated with the static CRD, Machine Config Pool, for the type of node you want to configure. Perform one of the following steps:

- a. Check current labels of the desired machine config pool.
For example:

```
$ oc get machineconfigpool --show-labels
```

Example output

```
NAME      CONFIG                                UPDATED  UPDATING  DEGRADED
LABELS
master    rendered-master-e05b81f5ca4db1d249a1bf32f9ec24fd  True    False
False     operator.machineconfiguration.openshift.io/required-for-upgrade=
worker    rendered-worker-f50e78e1bc06d8e82327763145bfcf62  True    False
False
```

- b. Add a custom label to the desired machine config pool.
For example:

```
$ oc label machineconfigpool worker custom-kubelet=enabled
```

2. Create a **kubeletconfig** custom resource (CR) for your configuration change, as demonstrated in the following sample configuration for a **custom-config** CR:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-config
spec:
  machineConfigPoolSelector:
    matchLabels:
```



```

    custom-kubelet: enabled
  kubeletConfig:
    podsPerCore: 10
    maxPods: 250
    systemReserved:
      cpu: 2000m
      memory: 1Gi
#...
```

where:

name

Assign a name to CR.

custom-kubelet

Specify the label to apply the configuration change, this is the label you added to the machine config pool.

kubeletConfig

Specify the new value(s) you want to change.

3. Create the CR object:

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

For example:

```
$ oc create -f master-kube-config.yaml
```

Most [Kubelet Configuration options](#) can be set by the user. The following options are not allowed to be overwritten:

- CgroupDriver
- ClusterDNS
- ClusterDomain
- StaticPodPath



NOTE

If a single node contains more than 50 images, pod scheduling might be imbalanced across nodes. This is because the list of images on a node is shortened to 50 by default. You can disable the image limit by editing the **KubeletConfig** object and setting the value of **nodeStatusMaxImages** to **-1**.

6.3.2. Configuring control plane nodes as schedulable

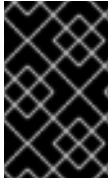
You can configure control plane nodes to be schedulable, meaning that new pods are allowed for placement on the control plane nodes. By default, control plane nodes are not schedulable.

You can set the control plane nodes to be schedulable, but must retain the compute nodes.

**NOTE**

You can deploy OpenShift Container Platform with no compute nodes on a bare-metal cluster. In this case, the control plane nodes are marked schedulable by default.

You can allow or disallow control plane nodes to be schedulable by configuring the **mastersSchedulable** field.

**IMPORTANT**

When you configure control plane nodes from the default unschedulable to schedulable, additional subscriptions are required. This is because control plane nodes then become compute nodes.

Procedure

1. Edit the **schedulers.config.openshift.io** resource.

```
$ oc edit schedulers.config.openshift.io cluster
```

2. Configure the **mastersSchedulable** field.

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  creationTimestamp: "2019-09-10T03:04:05Z"
  generation: 1
  name: cluster
  resourceVersion: "433"
  selfLink: /apis/config.openshift.io/v1/schedulers/cluster
  uid: a636d30a-d377-11e9-88d4-0a60097bee62
spec:
  mastersSchedulable: false
status: {}
#...
```

where:

mastersSchedulable

Set to **true** to allow control plane nodes to be schedulable, or **false** to disallow control plane nodes to be schedulable.

3. Save the file to apply the changes.

6.3.3. Setting SELinux booleans

OpenShift Container Platform allows you to enable and disable an SELinux boolean on a Red Hat Enterprise Linux CoreOS (RHCOS) node. The following procedure explains how to modify SELinux booleans on nodes using the Machine Config Operator (MCO). This procedure uses **container_manage_cgroup** as the example boolean. You can modify this value to whichever boolean you need.

Prerequisites

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

Procedure

1. Create a new YAML file with a **MachineConfig** object, displayed in the following example:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-worker-setsebool
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
      - contents: |
          [Unit]
          Description=Set SELinux booleans
          Before=kubelet.service

          [Service]
          Type=oneshot
          ExecStart=/sbin/setsebool container_manage_cgroup=on
          RemainAfterExit=true

          [Install]
          WantedBy=multi-user.target graphical.target
        enabled: true
        name: setsebool.service
#...
```

2. Create the new **MachineConfig** object by running the following command:

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



NOTE

Applying any changes to the **MachineConfig** object causes all affected nodes to gracefully reboot after the change is applied.

6.3.4. Adding kernel arguments to nodes

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

**WARNING**

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

Examples of kernel arguments you could set include:

- **nosmt**: Disables symmetric multithreading (SMT) in the kernel. Multithreading allows multiple logical threads for each CPU. You could consider **nosmt** in multi-tenant environments to reduce risks from potential cross-thread attacks. By disabling SMT, you essentially choose security over performance.
- **enforcing=0**: Configures Security Enhanced Linux (SELinux) to run in permissive mode. In permissive mode, the system acts as if SELinux is enforcing the loaded security policy, including labeling objects and emitting access denial entries in the logs, but it does not actually deny any operations. While not supported for production systems, permissive mode can be helpful for debugging.

**WARNING**

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

See [Kernel.org kernel parameters](https://kernel.org/doc/Documentation/kernel-parameters.txt) for a list and descriptions of kernel arguments.

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

- A set of machines to which you want to add the kernel argument. In this case, machines with a worker role.
- Kernel arguments that are appended to the end of the existing kernel arguments.
- A label that indicates where in the list of machine configs the change is applied.

Prerequisites

- You have **cluster-admin** privileges.
- Your cluster is running.

Procedure

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

```
$ oc get MachineConfig
```

Example output

NAME	GENERATEDBY	CONTROLLER
IGNITION	VERSION	AGE
00-master	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.5.0
33m		
00-worker	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.5.0
33m		
01-master-container-runtime	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.5.0		33m
01-master-kubelet	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.5.0		33m
01-worker-container-runtime	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.5.0		33m
01-worker-kubelet	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.5.0		33m
99-master-generated-registries	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.5.0		33m
99-master-ssh		3.2.0 40m
99-worker-generated-registries	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.5.0		33m
99-worker-ssh		3.2.0 40m
rendered-master-23e785de7587df95a4b517e0647e5ab7		
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.5.0	33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1		
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.5.0	33m

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

```

apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 05-worker-kernelarg-selinuxpermissive
spec:
  kernelArguments:
    - enforcing=0 1

```

where:

machineconfiguration.openshift.io/role

Applies the new kernel argument only to worker nodes.

name

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

kernelArguments

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

3. Create the new machine config:

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

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

```
$ oc get MachineConfig
```

Example output

NAME	GENERATEDBY	CONTROLLER
IGNITIONVERSION AGE		
00-master	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.5.0
33m		
00-worker	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	3.5.0
33m		
01-master-container-runtime	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.5.0 33m		
01-master-kubelet	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.5.0 33m		
01-worker-container-runtime	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.5.0 33m		
01-worker-kubelet	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.5.0 33m		
05-worker-kernelarg-selinuxpermissive		3.5.0 105s
99-master-generated-registries	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.5.0 33m		
99-master-ssh		3.2.0 40m
99-worker-generated-registries	52dd3ba6a9a527fc3ab42afac8d12b693534c8c9	
3.5.0 33m		
99-worker-ssh		3.2.0 40m
rendered-master-23e785de7587df95a4b517e0647e5ab7		
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.5.0		33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1		
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9 3.5.0		33m

5. Check the nodes:

```
$ oc get nodes
```

Example output

NAME	STATUS	ROLES	AGE	VERSION
ip-10-0-136-161.ec2.internal	Ready	worker	28m	v1.33.4
ip-10-0-136-243.ec2.internal	Ready	master	34m	v1.33.4
ip-10-0-141-105.ec2.internal	Ready,SchedulingDisabled	worker	28m	v1.33.4
ip-10-0-142-249.ec2.internal	Ready	master	34m	v1.33.4
ip-10-0-153-11.ec2.internal	Ready	worker	28m	v1.33.4
ip-10-0-153-150.ec2.internal	Ready	master	34m	v1.33.4

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

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

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

Example output

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

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

sh-4.2# exit
```

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

6.3.5. Allowing swap memory use on nodes

You can allow workloads on the cluster nodes to use swap memory.



IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).



NOTE

Swap memory support is available only for container-native virtualization (CNV) users or use cases.

To allow swap memory usage on your nodes, create a **kubeletconfig** custom resource (CR) to set the **failSwapOn** parameter to **false**.

Optionally, you can control swap memory usage by OpenShift Container Platform workloads on those nodes by setting the **swapBehavior** parameter to one of the following values:

- **NoSwap** prevents OpenShift Container Platform workloads from using swap memory.
- **LimitedSwap** allows OpenShift Container Platform workloads that fall under the Burstable QoS class to use swap memory.

Regardless of the **swapBehavior** setting, any workloads that are not managed by OpenShift Container Platform on that node can still use swap memory if the **failSwapOn** parameter is **false**.

Because the kubelet will not start in the presence of swap memory without this configuration, you must allow swap memory in OpenShift Container Platform before enabling swap memory on the nodes. If

there is no swap memory present on a node, enabling swap memory in OpenShift Container Platform has no effect.



WARNING

Using swap memory can negatively impact workload performance and out-of-resource handling. Do not enable swap memory on control plane nodes.

Prerequisites

- You have a running OpenShift Container Platform cluster that uses version 4.10 or later.
- Your cluster is configured to use cgroup v2. Swap memory is not supported on nodes in clusters that use cgroup v1.
- You are logged in to the cluster as a user with administrative privileges.
- You have enabled the **TechPreviewNoUpgrade** feature set on the cluster (see *Nodes → Working with clusters → Enabling features using feature gates*).



NOTE

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

Procedure

1. Apply a custom label to the machine config pool where you want to allow swap memory.

```
$ oc label machineconfigpool worker kubelet-swap=enabled
```

2. Create a custom resource (CR) to enable and configure swap settings.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: swap-config
spec:
  machineConfigPoolSelector:
    matchLabels:
      kubelet-swap: enabled
  kubeletConfig:
    failSwapOn: false
    memorySwap:
      swapBehavior: LimitedSwap
#...
```

where:

failSwapOn

Set to **false** to enable swap memory use on the associated nodes. Set to **true** to disable swap memory use.

swapBehavior

Optional: Specify the swap memory behavior for OpenShift Container Platform pods.

- **NoSwap**: OpenShift Container Platform pods cannot use swap. This is the default.
- **LimitedSwap**: OpenShift Container Platform pods of Burstable QoS class only are permitted to employ swap.

3. Enable swap memory on the nodes by setting the **swapaccount=1** kernel argument and configure swap memory as needed.

6.3.6. About configuring parallel container image pulls

To help control bandwidth issues, you can configure the number of workload images that can be pulled at the same time.

By default, the cluster pulls images in parallel, which allows multiple workloads to pull images at the same time. Pulling multiple images in parallel can improve workload start-up time because workloads can pull needed images without waiting for each other. However, pulling too many images at the same time can use excessive network bandwidth and cause latency issues throughout your cluster.

The default setting allows unlimited simultaneous image pulls. But, you can configure the maximum number of images that can be pulled in parallel. You can also force serial image pulling, which means that only one image can be pulled at a time.

To control the number of images that can be pulled simultaneously, use a kubelet configuration to set the **maxParallelImagePulls** to specify a limit. Additional image pulls above this limit are held until one of the current pulls is complete.

To force serial image pulls, use a kubelet configuration to set **serializeImagePulls** field to **true**.

6.3.6.1. Configuring parallel container image pulls

You can control the number of images that can be pulled by your workload simultaneously by using a kubelet configuration. You can set a maximum number of images that can be pulled or force workloads to pull images one at a time.

Prerequisites

- You have a running OpenShift Container Platform cluster.
- You are logged in to the cluster as a user with administrative privileges.

Procedure

1. Apply a custom label to the machine config pool where you want to configure parallel pulls by running a command similar to the following.

```
$ oc label machineconfigpool <mcp_name> parallel-pulls=set
```

2. Create a custom resource (CR) to configure parallel image pulling.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: parallel-image-pulls
# ...
spec:
  machineConfigPoolSelector:
    matchLabels:
      parallel-pulls: set
  kubeletConfig:
    serializeImagePulls: false
    maxParallelImagePulls: 3
# ...
```

where:

serializeImagePulls

Set to **false** to enable parallel image pulls. Set to **true** to force serial image pulling. The default is **false**.

maxParallelImagePulls

Specify the maximum number of images that can be pulled in parallel. Enter a number or set to **nil** to specify no limit. This field cannot be set if **SerializeImagePulls** is **true**. The default is **nil**.

3. Create the new machine config by running a command similar to the following:

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

Verification

1. Check the machine configs to see that a new one was added by running the following command:

```
$ oc get MachineConfig
```

NAME	IGNITIONVERSION	AGE	GENERATEDBYCONTROLLER
00-master	3.5.0	133m	70025364a114fc3067b2e82ce47fdb0149630e4b
00-worker	3.5.0	133m	70025364a114fc3067b2e82ce47fdb0149630e4b
# ...			
99-parallel-generated-kubelet	3.5.0	15s 1	70025364a114fc3067b2e82ce47fdb0149630e4b
# ...			
rendered-parallel-c634a80f644740974ceb40c054c79e50	70025364a114fc3067b2e82ce47fdb0149630e4b	3.5.0	10s 2

where:

99-parallel-generated-kubelet

The new machine config. In this example, the machine config is for the **parallel** custom machine config pool.

rendered-parallel-<sha_numnber>

The new rendered machine config. In this example, the machine config is for the **parallel** custom machine config pool.

2. Check to see that the nodes in the **parallel** machine config pool are being updated by running the following command:

```
$ oc get machineconfigpool
```

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT		
DEGRADEDMACHINECOUNT	AGE			
parallel	rendered-parallel-3904f0e69130d125b3b5ef0e981b1ce1	False	True	False
1	0	0	0	65m
master	rendered-master-7536834c197384f3734c348c1d957c18	True	False	False
3	3	3	0	140m
worker	rendered-worker-c634a80f644740974ceb40c054c79e50	True	False	False
2	2	2	0	140m

3. When the nodes are updated, verify that the parallel pull maximum was configured:
 - a. Open an **oc debug** session to a node by running a command similar to the following:

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

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

```
sh-5.1# chroot /host
```

- c. Examine the **kubelet.conf** file by running the following command:

```
sh-5.1# cat /etc/kubernetes/kubelet.conf | grep -i maxParallelImagePulls
```

```
maxParallelImagePulls: 3
```

6.3.7. Migrating control plane nodes from one RHOSP host to another manually

If control plane machine sets are not enabled on your cluster, you can run a script that moves a control plane node from one Red Hat OpenStack Platform (RHOSP) node to another.



NOTE

Control plane machine sets are not enabled on clusters that run on user-provisioned infrastructure.

For information about control plane machine sets, see "Managing control plane machines with control plane machine sets".

Prerequisites

- The environment variable **OS_CLOUD** refers to a **clouds** entry that has administrative credentials in a **clouds.yaml** file.
- The environment variable **KUBECONFIG** refers to a configuration that contains administrative OpenShift Container Platform credentials.

Procedure

- From a command line, run the following script:

```
#!/usr/bin/env bash

set -Eeuo pipefail

if [ $# -lt 1 ]; then
    echo "Usage: '$0 node_name'"
    exit 64
fi

# Check for admin OpenStack credentials
openstack server list --all-projects >/dev/null || { >&2 echo "The script needs OpenStack
admin credentials. Exiting"; exit 77; }

# Check for admin OpenShift credentials
oc adm top node >/dev/null || { >&2 echo "The script needs OpenShift admin credentials.
Exiting"; exit 77; }

set -x

declare -r node_name="$1"
declare server_id
server_id="$(openstack server list --all-projects -f value -c ID -c Name | grep "$node_name" |
cut -d' ' -f1)"
readonly server_id

# Drain the node
oc adm cordon "$node_name"
oc adm drain "$node_name" --delete-emptydir-data --ignore-daemonsets --force

# Power off the server
oc debug "node/${node_name}" -- chroot /host shutdown -h 1

# Verify the server is shut off
until openstack server show "$server_id" -f value -c status | grep -q 'SHUTOFF'; do sleep 5;
done

# Migrate the node
openstack server migrate --wait "$server_id"

# Resize the VM
openstack server resize confirm "$server_id"

# Wait for the resize confirm to finish
until openstack server show "$server_id" -f value -c status | grep -q 'SHUTOFF'; do sleep 5;
done
```

```

# Restart the VM
openstack server start "$server_id"

# Wait for the node to show up as Ready:
until oc get node "$node_name" | grep -q "^${node_name}[:space:]+\+Ready"; do sleep 5;
done

# Uncordon the node
oc adm uncordon "$node_name"

# Wait for cluster operators to stabilize
until oc get co -o go-template='status: {{ range .items }}{{ range .status.conditions }}{{ if eq
.type "Degraded" }}{{ if ne .status "False" }}DEGRADED{{ end }}{{ else if eq .type
"Progressing" }}{{ if ne .status "False" }}PROGRESSING{{ end }}{{ else if eq .type "Available" }}
{{ if ne .status "True" }}NOTAVAILABLE{{ end }}{{ end }}{{ end }}{{ end }}{{ end }}' | grep -qv '\
(DEGRADED\|PROGRESSING\|NOTAVAILABLE)'; do sleep 5; done

```

If the script completes, the control plane machine is migrated to a new RHOSP node.

6.3.8. Additional resources

- [Managing control plane machines with control plane machine sets](#)

6.4. ADDING WORKER NODES TO AN ON-PREMISE CLUSTER

For on-premise clusters, you can add worker nodes by using the [OpenShift 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 required configurations that are not specified during ISO generation are retrieved from the target cluster and applied to the new nodes.



NOTE

Machine or **BareMetalHost** resources are not automatically created after a node has been successfully added to the cluster.

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.

6.4.1. Supported platforms

The following platforms are supported for this method of adding nodes:

- **baremetal**
- **vsphere**
- **nutanix**
- **none**

6.4.2. Supported architectures

The following architecture combinations have been validated to work when adding worker nodes using this process:

- **amd64** worker nodes on **amd64** or **arm64** clusters
- **arm64** worker nodes on **amd64** or **arm64** clusters
- **s390x** worker nodes on **s390x** clusters
- **ppc64le** worker nodes on **ppc64le** clusters

6.4.3. Adding nodes to your cluster

You can add nodes with this method in the following two ways:

- Adding one or more nodes using a configuration file.
You can specify configurations for one or more nodes in the **nodes-config.yaml** file before running the **oc adm node-image create** command. This is useful if you want to add more than one node at a time, or if you are specifying complex configurations.
- Adding a single node using only command flags.
You can add a node by running the **oc adm node-image create** command with flags to specify your configurations. This is useful if you want to add only a single node at a time, and have only simple configurations to specify for that node.

6.4.3.1. Adding one or more nodes using a configuration file

You can add one or more nodes to your cluster by using the **nodes-config.yaml** file to specify configurations for the new nodes.

Prerequisites

- You have installed the OpenShift CLI (**oc**)
- You have installed the Rsync utility
- You have an active connection to your target cluster
- You have a kubeconfig file available

Procedure

1. Create a new YAML file that contains configurations for the nodes you are adding and is named **nodes-config.yaml**. You must provide a MAC address for each new node.
In the following example file, two new workers are described with an initial static network configuration:

Example nodes-config.yaml file

```
hosts:
- hostname: extra-worker-1
  rootDeviceHints:
    deviceName: /dev/sda
```

```

interfaces:
- macAddress: 00:00:00:00:00:00
  name: eth0
networkConfig:
  interfaces:
  - name: eth0
    type: ethernet
    state: up
    mac-address: 00:00:00:00:00:00
    ipv4:
      enabled: true
      address:
      - ip: 192.168.122.2
        prefix-length: 23
      dhcp: false
- hostname: extra-worker-2
  rootDeviceHints:
    deviceName: /dev/sda
  interfaces:
  - macAddress: 00:00:00:00:00:02
    name: eth0
  networkConfig:
    interfaces:
    - name: eth0
      type: ethernet
      state: up
      mac-address: 00:00:00:00:00:02
      ipv4:
        enabled: true
        address:
        - ip: 192.168.122.3
          prefix-length: 23
      dhcp: false

```

2. Generate the ISO image by running the following command:

```
$ oc adm node-image create nodes-config.yaml
```



IMPORTANT

In order for the **create** command to fetch a release image that matches the target cluster version, you must specify a valid pull secret. You can specify the pull secret either by using the **--registry-config** flag or by setting the **REGISTRY_AUTH_FILE** environment variable beforehand.



NOTE

If the directory of the **nodes-config.yaml** file is not specified by using the **--dir** flag, the tool looks for the file in the current directory.

3. Verify that a new **node.<arch>.iso** file is present in the asset directory. The asset directory is your current directory, unless you specified a different one when creating the ISO image.
4. Boot the selected node with the generated ISO image.

5. Track progress of the node creation by running the following command:

```
$ oc adm node-image monitor --ip-addresses <ip_addresses>
```

where:

<ip_addresses>

Specifies a list of the IP addresses of the nodes that are being added.



NOTE

If reverse DNS entries are not available for your node, the **oc adm node-image monitor** command skips checks for pending certificate signing requests (CSRs). If these checks are skipped, you must manually check for CSRs by running the **oc get csr** command.

6. Approve the CSRs by running the following command for each CSR:

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

6.4.3.2. Adding a node with command flags

You can add a single node to your cluster by using command flags to specify configurations for the new node.

Prerequisites

- You have installed the OpenShift CLI (**oc**)
- You have installed the Rsync utility
- You have an active connection to your target cluster
- You have a kubeconfig file available

Procedure

1. Generate the ISO image by running the following command. The MAC address must be specified using a command flag. See the "Cluster configuration reference" section for more flags that you can use with this command.

```
$ oc adm node-image create --mac-address=<mac_address>
```

where:

<mac_address>

Specifies the MAC address of the node that is being added.



IMPORTANT

In order for the **create** command to fetch a release image that matches the target cluster version, you must specify a valid pull secret. You can specify the pull secret either by using the **--registry-config** flag or by setting the **REGISTRY_AUTH_FILE** environment variable beforehand.

TIP

To see additional flags that can be used to configure your node, run the following **oc adm node-image create --help** command.

2. Verify that a new **node.<arch>.iso** file is present in the asset directory. The asset directory is your current directory, unless you specified a different one when creating the ISO image.
3. Boot the node with the generated ISO image.
4. Track progress of the node creation by running the following command:

```
$ oc adm node-image monitor --ip-addresses <ip_address>
```

where:

<ip_address>

Specifies a list of the IP addresses of the nodes that are being added.



NOTE

If reverse DNS entries are not available for your node, the **oc adm node-image monitor** command skips checks for pending certificate signing requests (CSRs). If these checks are skipped, you must manually check for CSRs by running the **oc get csr** command.

5. Approve the pending CSRs by running the following command for each CSR:

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

6.4.4. Cluster configuration reference

When creating the ISO image, configurations are retrieved from the target cluster and are applied to the new nodes. Any configurations for your cluster are applied to the nodes unless you override the configurations in either the **nodes-config.yaml** file or any flags you add to the **oc adm node-image create** command.

6.4.4.1. YAML file parameters

Configuration parameters that can be specified in the **nodes-config.yaml** file are described in the following table:

Table 6.2. nodes-config.yaml parameters

Parameter	Description	Values
<code>hosts:</code>	Host configuration.	An array of host configuration objects.
<code>hosts: hostname:</code>	Hostname. Overrides the hostname obtained from either the Dynamic Host Configuration Protocol (DHCP) or a reverse DNS lookup. Each host must have a unique hostname supplied by one of these methods, although configuring a hostname through this parameter is optional.	String.
<code>hosts: interfaces:</code>	Provides a table of the name and MAC address mappings for the interfaces on the host. If a NetworkConfig section is provided in the nodes-config.yaml file, this table must be included and the values must match the mappings provided in the NetworkConfig section.	An array of host configuration objects.
<code>hosts: interfaces: name:</code>	The name of an interface on the host.	String.
<code>hosts: interfaces: macAddress:</code>	The MAC address of an interface on the host.	A MAC address such as the following example: 00-B0-D0-63-C2-26 .
<code>hosts: rootDeviceHints:</code>	Enables provisioning of the Red Hat Enterprise Linux CoreOS (RHCOS) image to a particular device. The node-adding tool examines the devices in the order it discovers them, and compares the discovered values with the hint values. It uses the first discovered device that matches the hint value.	A dictionary of key-value pairs. For more information, see "Root device hints" in the "Setting up the environment for an OpenShift installation" page.
<code>hosts: rootDeviceHints: deviceName:</code>	The name of the device the RHCOS image is provisioned to.	String.

Parameter	Description	Values
hosts: networkConfig:	The host network definition. The configuration must match the Host Network Management API defined in the nmstate documentation .	A dictionary of host network configuration objects.
cpuArchitecture:	Optional. Specifies the architecture of the nodes you are adding. This parameter allows you to override the default value from the cluster when required.	String.
sshKey:	Optional. The file containing the SSH key to authenticate access to your cluster machines.	String.
bootArtifactsBaseURL:	Optional. Specifies the URL of the server to upload Preboot Execution Environment (PXE) assets to when you are generating an iPXE script. You must also set the --pxe flag to generate PXE assets instead of an ISO image.	String.

6.4.4.2. Command flag options

You can use command flags with the **oc adm node-image create** command to configure the nodes you are creating.

The following table describes command flags that are not limited to the single-node use case:

Table 6.3. General command flags

Flag	Description	Values
--certificate-authority	The path to a certificate authority bundle to use when communicating with the managed container image registries. If the -insecure flag is used, this flag is ignored.	String

Flag	Description	Values
--dir	The path containing the configuration file, if provided. This path is also used to store the generated artifacts.	String
--insecure	Allows push and pull operations to registries to be made over HTTP.	Boolean
-o, --output-name	The name of the generated output image.	String
p, --pxe	Generates Preboot Execution Environment (PXE) assets instead of a bootable ISO file. When this flag is set, you can also use the bootArtifactsBaseURL parameter in the nodes-config.yaml file to specify URL of the server you will upload PXE assets to.	Boolean
-a, --registry-config	The path to your registry credentials. Alternatively, you can specify the REGISTRY_AUTH_FILE environment variable. The default paths are \${XDG_RUNTIME_DIR}/containers/auth.json, /run/containers/\${UID}/auth.json, \${XDG_CONFIG_HOME}/containers/auth.json, \${DOCKER_CONFIG}, ~/.docker/config.json, ~/.dockercfg. The order can be changed through the deprecated REGISTRY_AUTH_PREFERENCE environment variable to a "docker" value, in order to prioritize Docker credentials over Podman.	String

Flag	Description	Values
-r, --report	Generates a report of the node creation process regardless of whether the process is successful or not. If you do not specify this flag, reports are generated only in cases of failure.	Boolean
--skip-verification	An option to skip verifying the integrity of the retrieved content. This is not recommended, but might be necessary when importing images from older image registries. Bypass verification only if the registry is known to be trustworthy.	Boolean

The following table describes command flags that can be used only when creating a single node:

Table 6.4. Single-node only command flags

Flag	Description	Values
-c, --cpu-architecture	The CPU architecture to be used to install the node. This flag can be used to create only a single node, and the --mac-address flag must be defined.	String
--hostname	The hostname to be set for the node. This flag can be used to create only a single node, and the --mac-address flag must be defined.	String
-m, --mac-address	The MAC address used to identify the host to apply configurations to. This flag can be used to create only a single node, and the --mac-address flag must be defined.	String
--network-config-path	The path to a YAML file containing NMState configurations to be applied to the node. This flag can be used to create only a single node, and the --mac-address flag must be defined.	String

Flag	Description	Values
--root-device-hint	A hint for specifying the storage location for the image root filesystem. The accepted format is <hint_name>:<value> . This flag can be used to create only a single node, and the --mac-address flag must be defined.	String
-k, --ssh-key-path	The path to the SSH key used to access the node. This flag can be used to create only a single node, and the --mac-address flag must be defined.	String

Additional resources

- [Root device hints](#)

6.5. MANAGING THE MAXIMUM NUMBER OF PODS PER NODE

In OpenShift Container Platform, you can configure the number of pods that can run on a node based on the number of processor cores on the node, a hard limit or both. If you use both options, the lower of the two limits the number of pods on a node.

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
```

6.5.1. 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 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 1
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" 2
  kubeletConfig:
    podsPerCore: 10 3
    maxPods: 250 4
#...
```

- 1** Assign a name to CR.
- 2** Specify the label from the machine config pool.
- 3** Specify the number of pods the node can run based on the number of processor cores on the node.
- 4** Specify the number of pods the node can run to a fixed value, regardless of the properties of the node.

**NOTE**

Setting **podsPerCore** to **0** disables this limit.

In the above example, the default value for **podsPerCore** is **10** and the default value for **maxPods** is **250**. This means that unless the node has 25 cores or more, by default, **podsPerCore** will be the limiting factor.

2. Run the following command to create the CR:

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

Verification

1. List the **MachineConfigPool** CRDs to see if the change is applied. The **UPDATING** column reports **True** if the change is picked up by the Machine Config Controller:

■


```
$ oc get machineconfigpools
```

Example output

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
master	master-9cc2c72f205e103bb534	False	False	False
worker	worker-8cecd1236b33ee3f8a5e	False	True	False

Once the change is complete, the **UPDATED** column reports **True**.

```
$ oc get machineconfigpools
```

Example output

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
master	master-9cc2c72f205e103bb534	False	True	False
worker	worker-8cecd1236b33ee3f8a5e	True	False	False

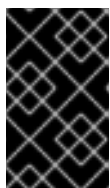
6.6. REPLACING A FAILED BARE-METAL CONTROL PLANE NODE WITHOUT BMC CREDENTIALS

If a control plane node on your bare-metal cluster has failed and cannot be recovered, but you installed your cluster without providing baseboard management controller (BMC) credentials, you must take extra steps in order to replace the failed node with a new one.

6.6.1. Prerequisites

- You have identified the unhealthy bare metal etcd member.
- You have verified that either the machine is not running or the node is not ready.
- You have access to the cluster as a user with the **cluster-admin** role.
- You have taken an [etcd backup](#) in case you encounter any issues.
- You have downloaded and installed the [coreos-installer CLI](#).
- Your cluster does not have a control plane **machineset**. You can check for **machinesets** by running the following command:

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



IMPORTANT

There should be only one or more **machinesets** for the workers. If **controlplanemachinesets** exists for the control plane, do not use this procedure.

6.6.2. Removing the unhealthy etcd member

Begin removing the failed control plane node by first removing the unhealthy etcd member.

Procedure

- 1. List etcd pods by running the following command and make note of a pod that is not on the affected node:

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd -o wide
```

Example output

```
etcd-openshift-control-plane-0 5/5 Running 11 3h56m 192.168.10.9 openshift-
control-plane-0 <none> <none>
etcd-openshift-control-plane-1 5/5 Running 0 3h54m 192.168.10.10 openshift-
control-plane-1 <none> <none>
etcd-openshift-control-plane-2 5/5 Running 0 3h58m 192.168.10.11 openshift-
control-plane-2 <none> <none>
```

- 2. Connect to a running etcd container by running the following command:

```
$ oc rsh -n openshift-etcd <etcd_pod>
```

Replace **<etcd_pod>** with the name of an etcd pod associated with one of the healthy nodes.

Example command

```
$ oc rsh -n openshift-etcd etcd-openshift-control-plane-0
```

- 3. View the etcd member list by running the following command. Make note of the ID and the name of the unhealthy etcd member because these values are required later.

```
sh-4.2# etcdctl member list -w table
```

Example output

```
+-----+-----+-----+-----+-----+
+-----+
| ID | STATUS | NAME | PEER ADDRS | CLIENT ADDRS |
+-----+-----+-----+-----+-----+
+-----+
| 6fc1e7c9db35841d | started | openshift-control-plane-2 | https://10.0.131.183:2380 | https://10.0.131.183:2379 |
| 757b6793e2408b6c | started | openshift-control-plane-1 | https://10.0.164.97:2380 | https://10.0.164.97:2379 |
| ca8c2990a0aa29d1 | started | openshift-control-plane-0 | https://10.0.154.204:2380 | https://10.0.154.204:2379 |
+-----+-----+-----+-----+-----+
+-----+
```



IMPORTANT

The **etcdctl endpoint health** command will list the removed member until the replacement is complete and the new member is added.

- Remove the unhealthy etcd member by running the following command:

```
sh-4.2# etcdctl member remove <unhealthy_member_id>
```

Replace **<unhealthy_member_id>** with the ID of the etcd member on the unhealthy node.

Example command

```
sh-4.2# etcdctl member remove 6fc1e7c9db35841d
```

Example output

```
Member 6fc1e7c9db35841d removed from cluster b23536c33f2cdd1b
```

- View the member list again by running the following command and verify that the member was removed:

```
sh-4.2# etcdctl member list -w table
```

Example output

```
+-----+-----+-----+-----+-----+
+-----+
| ID      | STATUS | NAME                | PEER ADDRS      | CLIENT
ADDRS    |
+-----+-----+-----+-----+-----+
+-----+
| 757b6793e2408b6c | started | openshift-control-plane-1 | https://10.0.164.97:2380 |
https://10.0.164.97:2379 |
| ca8c2990a0aa29d1 | started | openshift-control-plane-0 | https://10.0.154.204:2380 |
https://10.0.154.204:2379 |
+-----+-----+-----+-----+-----+
+-----+
```



IMPORTANT

After you remove the member, the cluster might be unreachable for a short time while the remaining etcd instances reboot.

- Exit the rsh session into the etcd pod by running the following command:

```
sh-4.2# exit
```

- Turn off the etcd quorum guard by running the following command:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"unsupportedConfigOverrides": {"useUnsupportedUnsafeNonHANonProductionUnstableEtcd": true}}}'
```

This command ensures that you can successfully re-create secrets and roll out the static pods.

- List the secrets for the removed, unhealthy etcd member by running the following command:

```
$ oc get secrets -n openshift-etcd | grep <node_name>
```

Replace **<node_name>** with the name of the failed node whose etcd member you removed.

Example command

```
$ oc get secrets -n openshift-etcd | grep openshift-control-plane-2
```

Example output

```
etcd-peer-openshift-control-plane-2      kubernetes.io/tls  2  134m
etcd-serving-metrics-openshift-control-plane-2 kubernetes.io/tls  2  134m
etcd-serving-openshift-control-plane-2    kubernetes.io/tls  2  134m
```

9. Delete the secrets associated with the affected node that was removed:

- a. Delete the peer secret by running the following command:

```
$ oc delete secret -n openshift-etcd etcd-peer-<node_name>
```

Replace **<node_name>** with the name of the affected node.

- b. Delete the serving secret by running the following command:

```
$ oc delete secret -n openshift-etcd etcd-serving-<node_name>
```

Replace **<node_name>** with the name of the affected node.

- c. Delete the metrics secret by running the following command:

```
$ oc delete secret -n openshift-etcd etcd-serving-metrics-<node_name> 1
```

Replace **<node_name>** with the name of the affected node.

6.6.3. Deleting the machine of the unhealthy etcd member

Finish removing the failed control plane node by deleting the machine of the unhealthy etcd member.

Procedure

1. Ensure that the Bare Metal Operator is available by running the following command:

```
$ oc get clusteroperator baremetal
```

Example output

```
NAME      VERSION AVAILABLE PROGRESSING DEGRADED SINCE  MESSAGE
baremetal 4.20.0   True      False      False   3d15h
```

2. Save the **BareMetalHost** object of the affected node to a file for later use by running the following command:

```
$ oc get -n openshift-machine-api bmh <node_name> -o yaml > bmh_affected.yaml
```

Replace **<node_name>** with the name of the affected node, which usually matches the associated **BareMetalHost** name.

3. View the YAML file of the saved **BareMetalHost** object by running the following command, and ensure the content is correct:

```
$ cat bmh_affected.yaml
```

4. Remove the affected **BareMetalHost** object by running the following command:

```
$ oc delete -n openshift-machine-api bmh <node_name>
```

Replace **<node_name>** with the name of the affected node.

5. List all machines by running the following command and identify the machine associated with the affected node:

```
$ oc get machines -n openshift-machine-api -o wide
```

Example output

```
NAME                                PHASE  TYPE  REGION  ZONE  AGE   NODE
PROVIDERID                                STATE
examplecluster-control-plane-0 Running                3h11m openshift-control-plane-0
baremetalhost:///openshift-machine-api/openshift-control-plane-0/da1ebe11-3ff2-41c5-b099-
0aa41222964e externally provisioned
examplecluster-control-plane-1 Running                3h11m openshift-control-plane-1
baremetalhost:///openshift-machine-api/openshift-control-plane-1/d9f9acbc-329c-475e-8d81-
03b20280a3e1 externally provisioned
examplecluster-control-plane-2 Running                3h11m openshift-control-plane-2
baremetalhost:///openshift-machine-api/openshift-control-plane-2/3354bdac-61d8-410f-be5b-
6a395b056135 externally provisioned
examplecluster-compute-0           Running                165m openshift-compute-0
baremetalhost:///openshift-machine-api/openshift-compute-0/3d685b81-7410-4bb3-80ec-
13a31858241f    provisioned
examplecluster-compute-1           Running                165m openshift-compute-1
baremetalhost:///openshift-machine-api/openshift-compute-1/0fdae6eb-2066-4241-91dc-
e7ea72ab13b9    provisioned
```

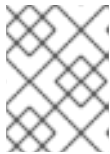
6. Delete the machine of the unhealthy member by running the following command:

```
$ oc delete machine -n openshift-machine-api <machine_name>
```

Replace **<machine_name>** with the machine name associated with the affected node.

Example command

```
$ oc delete machine -n openshift-machine-api examplecluster-control-plane-2
```



NOTE

After you remove the **BareMetalHost** and **Machine** objects, the machine controller automatically deletes the **Node** object.

- 7. If deletion of the machine is delayed for any reason or the command is obstructed and delayed, force deletion by removing the machine object finalizer field.



WARNING

Do not interrupt machine deletion by pressing **Ctrl+c**. You must allow the command to proceed to completion. Open a new terminal window to edit and delete the finalizer fields.

- a. On a new terminal window, edit the machine configuration by running the following command:

```
$ oc edit machine -n openshift-machine-api examplecluster-control-plane-2
```

- b. Delete the following fields in the **Machine** custom resource, and then save the updated file:

```
finalizers:
- machine.machine.openshift.io
```

Example output

```
machine.machine.openshift.io/examplecluster-control-plane-2 edited
```

6.6.4. Verifying that the failed node was deleted

Before proceeding to create a replacement control plane node, verify that the failed node was successfully deleted.

Procedure

- 1. Verify that the machine was deleted by running the following command:

```
$ oc get machines -n openshift-machine-api -o wide
```

Example output

NAME	PHASE	TYPE	REGION	ZONE	AGE	NODE
PROVIDERID						STATE
examplecluster-control-plane-0	Running				3h11m	openshift-control-plane-0
baremetalhost:///openshift-machine-api/openshift-control-plane-0/da1ebe11-3ff2-41c5-b099-0aa41222964e	externally provisioned					
examplecluster-control-plane-1	Running				3h11m	openshift-control-plane-1

```

baremetalhost:///openshift-machine-api/openshift-control-plane-1/d9f9acbc-329c-475e-8d81-
03b20280a3e1 externally provisioned
examplecluster-compute-0 Running 165m openshift-compute-0
baremetalhost:///openshift-machine-api/openshift-compute-0/3d685b81-7410-4bb3-80ec-
13a31858241f provisioned
examplecluster-compute-1 Running 165m openshift-compute-1
baremetalhost:///openshift-machine-api/openshift-compute-1/0fdae6eb-2066-4241-91dc-
e7ea72ab13b9 provisioned

```

2. Verify that the node has been deleted by running the following command:

```
$ oc get nodes
```

Example output

```

NAME                STATUS ROLES  AGE  VERSION
openshift-control-plane-0 Ready master 3h24m v1.33.4
openshift-control-plane-1 Ready master 3h24m v1.33.4
openshift-compute-0   Ready worker 176m v1.33.4
openshift-compute-1   Ready worker 176m v1.33.4

```

3. Wait for all of the cluster Operators to complete rolling out changes. Run the following command to monitor the progress:

```
$ watch oc get co
```

6.6.5. Creating the new control plane node

Begin creating the new control plane node by creating a **BareMetalHost** object and node.

Procedure

1. Edit the **bmh_affected.yaml** file that you previously saved:
 - a. Remove the following metadata items from the file:
 - **creationTimestamp**
 - **generation**
 - **resourceVersion**
 - **uid**
 - b. Remove the **status** section of the file.

The resulting file should resemble the following example:

Example **bmh_affected.yaml** file

```

apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  labels:

```

```

    installer.openshift.io/role: control-plane
    name: openshift-control-plane-2
    namespace: openshift-machine-api
  spec:
    automatedCleaningMode: disabled
  bmc:
    address:
    credentialsName:
    disableCertificateVerification: true
    bootMACAddress: ab:cd:ef:ab:cd:ef
    bootMode: UEFI
    externallyProvisioned: true
    online: true
  rootDeviceHints:
    deviceName: /dev/disk/by-path/pci-0000:04:00.0-nvme-1
  userData:
    name: master-user-data-managed
    namespace: openshift-machine-api

```

2. Create the **BareMetalHost** object using the **bmh_affected.yaml** file by running the following command:

```
$ oc create -f bmh_affected.yaml
```

The following warning is expected upon creation of the **BareMetalHost** object:

```
Warning: metadata.finalizers: "baremetalhost.metal3.io": prefer a domain-qualified finalizer
name to avoid accidental conflicts with other finalizer writers
```

3. Extract the control plane ignition secret by running the following command:

```
$ oc extract secret/master-user-data-managed \
  -n openshift-machine-api \
  --keys=userData \
  --to=- \
  | sed '/^userData/d' > new_controlplane.ign
```

This command also removes the starting **userData** line of the ignition secret.

4. Create an Nmstate YAML file titled **new_controlplane_nmstate.yaml** for the new node's network configuration, using the following example for reference:

Example Nmstate YAML file

```

interfaces:
- name: eno1
  type: ethernet
  state: up
  mac-address: "ab:cd:ef:01:02:03"
  ipv4:
    enabled: true
    address:
    - ip: 192.168.20.11
      prefix-length: 24

```



```

    dhcp: false
    ipv6:
      enabled: false
  dns-resolver:
    config:
      search:
        - iso.sterling.home
      server:
        - 192.168.20.8
  routes:
    config:
      - destination: 0.0.0.0/0
        metric: 100
        next-hop-address: 192.168.20.1
        next-hop-interface: eno1
        table-id: 254

```



NOTE

If you installed your cluster using the Agent-based Installer, you can use the failed node's **networkConfig** section in the **agent-config.yaml** file from the original cluster deployment as a starting point for the new control plane node's Nmstate file. For example, the following command extracts the **networkConfig** section for the first control plane node:

```
$ cat agent-config-iso.yaml | yq .hosts[0].networkConfig >
new_controlplane_nmstate.yaml
```

5. Create the customized Red Hat Enterprise Linux CoreOS (RHCOS) live ISO by running the following command:

```
$ coreos-installer iso customize rhcos-live.86_64.iso \
  --dest-ignition new_controlplane.ign \
  --network-nmstate new_controlplane_nmstate.yaml \
  --dest-device /dev/disk/by-path/<device_path> \
  -f
```

Replace **<device_path>** with the path to the target device on which the ISO will be generated.

6. Boot the new control plane node with the customized RHCOS live ISO.
7. Approve the Certificate Signing Requests (CSR) to join the new node to the cluster.

6.6.6. Linking the node, bare metal host, and machine together

Continue creating the new control plane node by creating a machine and then linking it with the new **BareMetalHost** object and node.

Procedure

1. Get the **providerID** for control plane nodes by running the following command:

```
$ oc get -n openshift-machine-api baremetalhost -l installer.openshift.io/role=control-plane -
ojson | jq -r '.items[] | "baremetalhost:///openshift-machine-api/" + .metadata.name + "/" +
.metadata.uid'
```

Example output

```
baremetalhost:///openshift-machine-api/master-00/6214c5cf-c798-4168-8c78-1ff1a3cd2cb4
baremetalhost:///openshift-machine-api/master-01/58fb60bd-b2a6-4ff3-a88d-208c33abf954
baremetalhost:///openshift-machine-api/master-02/dc5a94f3-625b-43f6-ab5a-7cc4fc79f105
```

2. Get cluster information for labels by running the following command:

```
$ oc get machine -n openshift-machine-api \
-l machine.openshift.io/cluster-api-machine-role=master \
-L machine.openshift.io/cluster-api-cluster
```

Example output

NAME	PHASE	TYPE	REGION	ZONE	AGE	CLUSTER-API-CLUSTER
ci-op-jcp3s7wx-ng5sd-master-0	Running			10h	ci-op-jcp3s7wx-ng5sd	
ci-op-jcp3s7wx-ng5sd-master-1	Running			10h	ci-op-jcp3s7wx-ng5sd	
ci-op-jcp3s7wx-ng5sd-master-2	Running			10h	ci-op-jcp3s7wx-ng5sd	

3. Create a **Machine** object for the new control plane node by creating a yaml file similar to the following:

```
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  annotations:
    metal3.io/BareMetalHost: openshift-machine-api/<new_control_plane_machine> ❶
  finalizers:
    - machine.machine.openshift.io
  labels:
    machine.openshift.io/cluster-api-cluster: <cluster_api_cluster> ❷
    machine.openshift.io/cluster-api-machine-role: master
    machine.openshift.io/cluster-api-machine-type: master
  name: <new_control_plane_machine> ❸
  namespace: openshift-machine-api
spec:
  metadata: {}
  providerID: <provider_id> ❹
  providerSpec:
    value:
      apiVersion: baremetal.cluster.k8s.io/v1alpha1
      hostSelector: {}
      image:
        checksum: ""
        url: ""
      kind: BareMetalMachineProviderSpec
      userData:
        name: master-user-data-managed
```

where:

<new_control_plane_machine>

Specifies the name of the new machine, which can be the same as the previously deleted machine name.

<cluster_api_cluster>

Specifies the **CLUSTER-API-CLUSTER** value for the other control plane machines, shown in the output of the previous step.

<provider_id>

Specifies the **providerID** value of the new bare metal host, shown in the output of an earlier step.

The following warning is expected:

Warning: metadata.finalizers: "machine.machine.openshift.io": prefer a domain-qualified finalizer name to avoid accidental conflicts with other finalizer writers

4. Link the new control plane node and **Machine** object to the **BareMetalHost** object by performing the following steps in a single bash shell session:

- a. Define the **NEW_NODE_NAME** variable by running the following command:

```
$ NEW_NODE_NAME=<new_node_name>
```

Replace **<new_node_name>** with the name of the new control plane node.

- b. Define the **NEW_MACHINE_NAME** variable by running the following command:

```
$ NEW_MACHINE_NAME=<new_machine_name>
```

Replace **<new_machine_name>** with the name of the new machine.

- c. Define the **BMH_UID** by running the following commands to extract it from the new node's **BareMetalHost** object:

```
$ BMH_UID=$(oc get -n openshift-machine-api bmh $NEW_NODE_NAME -ojson | jq -r .metadata.uid)
```

```
$ echo $BMH_UID
```

- d. Patch the **consumerRef** object into the bare metal host by running the following command:

```
$ oc patch -n openshift-machine-api bmh $NEW_NODE_NAME --type merge --patch
'{"spec":{"consumerRef":
{"apiVersion":"machine.openshift.io/v1beta1","kind":"Machine","name":"$NEW_MACHINE_
NAME","namespace":"openshift-machine-api"}}}'
```

- e. Patch the **providerID** value into the new node by running the following command:

```
$ oc patch node $NEW_NODE_NAME --type merge --patch '{"spec":
{"providerID":"baremetalhost:///openshift-machine-
api/$NEW_NODE_NAME/$BMH_UID"}}'
```

- f. Review the **providerID** values by running the following command:

```
$ oc get node -l node-role.kubernetes.io/control-plane -ojson | jq -r '.items[] |
.metadata.name + " " + .spec.providerID'
```

5. Set the **BareMetalHost** object's **poweredOn** status to **true** by running the following command:

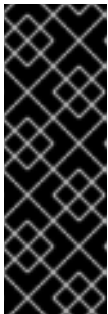
```
$ oc patch -n openshift-machine-api bmh $NEW_NODE_NAME --subresource status --type
json -p '[{"op":"replace","path":"/status/poweredOn","value":true}]'
```

6. Review the **BareMetalHost** object's **poweredOn** status by running the following command:

```
$ oc get bmh -n openshift-machine-api -ojson | jq -r '.items[] | .metadata.name + "
PoweredOn:" + (.status.poweredOn | tostring)'
```

7. Review the **BareMetalHost** object's provisioning state by running the following command:

```
$ oc get bmh -n openshift-machine-api -ojson | jq -r '.items[] | .metadata.name + "
ProvisioningState:" + .status.provisioning.state'
```



IMPORTANT

If the provisioning state is not **unmanaged**, change the provisioning state by running the following command:

```
$ oc patch -n openshift-machine-api bmh $NEW_NODE_NAME --subresource
status --type json -p
'[{"op":"replace","path":"/status/provisioning/state","value":"unmanaged"}]'
```

8. Set the machine's state to **Provisioned** by running the following command:

```
$ oc patch -n openshift-machine-api machines $NEW_MACHINE_NAME -n openshift-
machine-api --subresource status --type json -p
'[{"op":"replace","path":"/status/phase","value":"Provisioned"}]'
```

6.6.7. Adding the new etcd member

Finish adding the new control plane node by adding the new etcd member to the cluster.

Procedure

1. Add the new etcd member to the cluster by performing the following steps in a single bash shell session:
 - a. Find the IP of the new control plane node by running the following command:

```
$ oc get nodes -owide -l node-role.kubernetes.io/control-plane
```

Make note of the node's IP address for later use.

- b. List the etcd pods by running the following command:

```
$ oc get -n openshift-etcd pods -l k8s-app=etcd -o wide
```

- c. Connect to one of the running etcd pods by running the following command. The etcd pod on the new node should be in a **CrashLoopBackOff** state.

```
$ oc rsh -n openshift-etcd <running_pod>
```

Replace **<running_pod>** with the name of a running pod shown in the previous step.

- d. View the etcd member list by running the following command:

```
sh-4.2# etcdctl member list -w table
```

- e. Add the new control plane etcd member by running the following command:

```
sh-4.2# etcdctl member add <new_node> --peer-urls="https://<ip_address>:2380"
```

where:

<new_node>

Specifies the name of the new control plane node

<ip_address>

Specifies the IP address of the new node.

- f. Exit the rsh shell by running the following command:

```
sh-4.2# exit
```

2. Force an etcd redeployment by running the following command:

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

3. Turn the etcd quorum guard back on by running the following command:

```
$ oc patch etcd/cluster --type=merge -p '{ "spec": { "unsupportedConfigOverrides": null } }
```

4. Monitor the cluster Operator rollout by running the following command:

```
$ watch oc get co
```

6.6.8. Additional resources

- [Replacing an unhealthy bare metal etcd member whose machine is not running or whose node is not ready](#)
- [Replacing an unhealthy etcd member whose etcd pod is crashlooping](#)

- [BareMetalHost reference is missing after adding a host to OpenShift Assisted Installer cluster \(Red Hat KCS article\)](#)
- [How to retrieve Master or Worker Ignition Configuration from OpenShift Container Platform 4? \(Red Hat KCS article\)](#)

6.7. USING THE NODE TUNING OPERATOR

Learn about the Node Tuning Operator and how you can use it to manage node-level tuning by orchestrating the tuned daemon.

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.

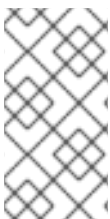
Node-level settings applied by the containerized TuneD daemon are rolled back on an event that triggers a profile change or when the containerized TuneD daemon is terminated gracefully by receiving and handling a termination signal.

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.

6.7.1. 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.

6.7.2. Custom tuning specification

The custom resource (CR) for the Operator has two major sections. The first section, **profile:**, is a list of TuneD profiles and their names. The second, **recommend:**, defines the profile selection logic.

Multiple custom tuning specifications can co-exist as multiple CRs in the Operator's namespace. The existence of new CRs or the deletion of old CRs is detected by the Operator. All existing custom tuning specifications are merged and appropriate objects for the containerized TuneD daemons are updated.

Management state

The Operator Management state is set by adjusting the default Tuned CR. By default, the Operator is in the Managed state and the **spec.managementState** field is not present in the default Tuned CR. Valid values for the Operator Management state are as follows:

- Managed: the Operator will update its operands as configuration resources are updated
- Unmanaged: the Operator will ignore changes to the configuration resources
- Removed: the Operator will remove its operands and resources the Operator provisioned

Profile data

The **profile:** section lists TuneD profiles and their names.

```
profile:
- name: tuned_profile_1
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_1 profile

    [sysctl]
    net.ipv4.ip_forward=1
```

```
# ... other sysctl's or other TuneD daemon plugins supported by the containerized TuneD

# ...

- name: tuned_profile_n
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_n profile

    # tuned_profile_n profile settings
```

Recommended profiles

The **profile:** selection logic is defined by the **recommend:** section of the CR. The **recommend:** section is a list of items to recommend the profiles based on a selection criteria.

```
recommend:
<recommend-item-1>
# ...
<recommend-item-n>
```

The individual items of the list:

```
- machineConfigLabels: ❶
  <mcLabels> ❷
  match: ❸
  <match> ❹
  priority: <priority> ❺
  profile: <tuned_profile_name> ❻
  operand: ❼
  debug: <bool> ❽
  tunedConfig:
    reapply_sysctl: <bool> ❾
```

❶ ❶ ❸ ❶ Optional.

❷ ❷ A dictionary of key/value **MachineConfig** labels. The keys must be unique.

❹ ❸ If omitted, profile match is assumed unless a profile with a higher priority matches first or **machineConfigLabels** is set.

❹ An optional list.

❺ Profile ordering priority. Lower numbers mean higher priority (**0** is the highest priority).

❻ A TuneD profile to apply on a match. For example **tuned_profile_1**.

❼ Optional operand configuration.

❽ Turn debugging on or off for the TuneD daemon. Options are **true** for on or **false** for off. The default is **false**.

❾ Turn **reapply_sysctl** functionality on or off for the TuneD daemon. Options are **true** for on and **false** for off.

raise TOR OTT.

<match> is an optional list recursively defined as follows:

```
- label: <label_name> 1
  value: <label_value> 2
  type: <label_type> 3
  <match> 4
```

- 1 Node or pod label name.
- 2 Optional node or pod label value. If omitted, the presence of **<label_name>** is enough to match.
- 3 Optional object type (**node** or **pod**). If omitted, **node** is assumed.
- 4 An optional **<match>** list.

If **<match>** is not omitted, all nested **<match>** sections must also evaluate to **true**. Otherwise, **false** is assumed and the profile with the respective **<match>** section will not be applied or recommended. Therefore, the nesting (child **<match>** sections) works as logical AND operator. Conversely, if any item of the **<match>** list matches, the entire **<match>** list evaluates to **true**. Therefore, the list acts as logical OR operator.

If **machineConfigLabels** is defined, machine config pool based matching is turned on for the given **recommend:** list item. **<mcLabels>** specifies the labels for a machine config. The machine config is created automatically to apply host settings, such as kernel boot parameters, for the profile **<tuned_profile_name>**. This involves finding all machine config pools with machine config selector matching **<mcLabels>** and setting the profile **<tuned_profile_name>** on all nodes that are assigned the found machine config pools. To target nodes that have both master and worker roles, you must use the master role.

The list items **match** and **machineConfigLabels** are connected by the logical OR operator. The **match** item is evaluated first in a short-circuit manner. Therefore, if it evaluates to **true**, the **machineConfigLabels** item is not considered.



IMPORTANT

When using machine config pool based matching, it is advised to group nodes with the same hardware configuration into the same machine config pool. Not following this practice might result in TuneD operands calculating conflicting kernel parameters for two or more nodes sharing the same machine config pool.

Example: Node or pod label based matching

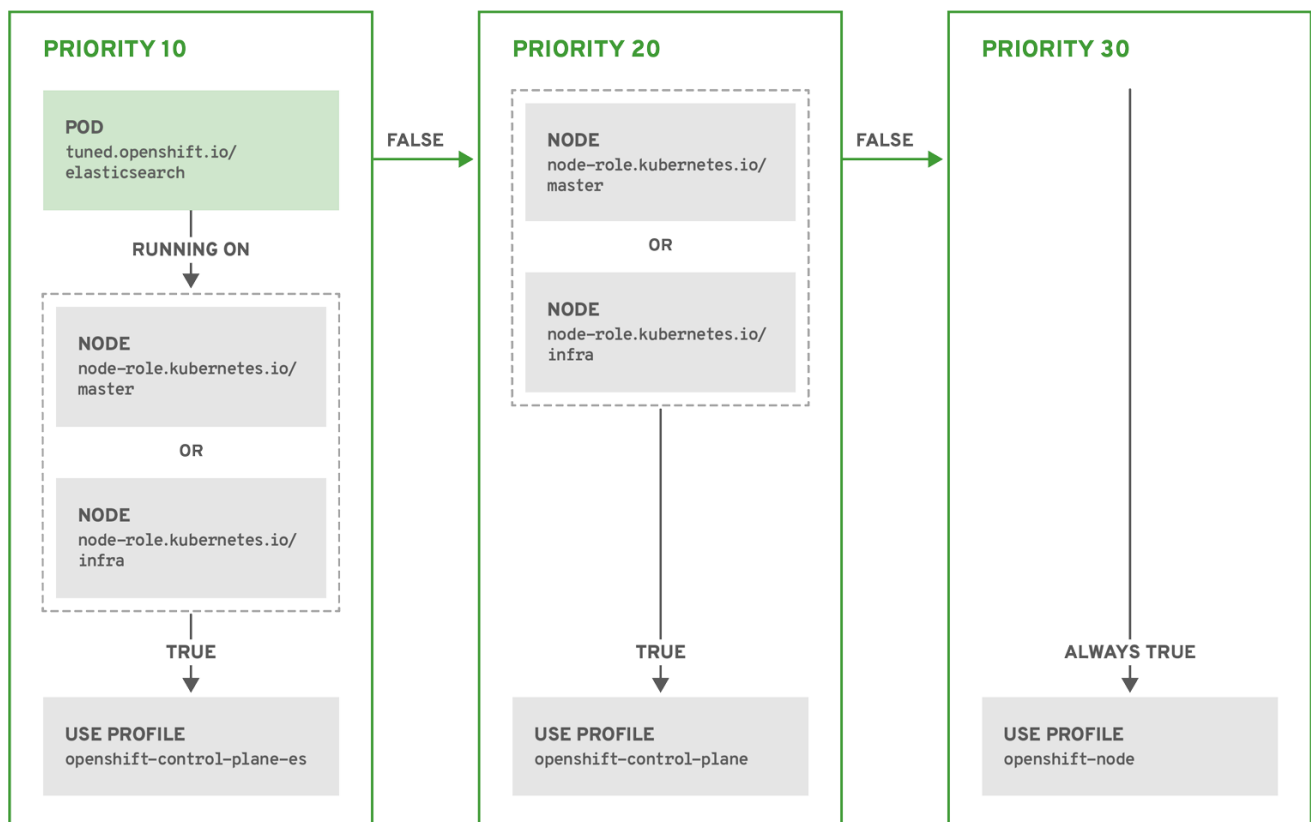
```
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
      - label: node-role.kubernetes.io/master
      - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
```

- match:
 - label: node-role.kubernetes.io/master
 - label: node-role.kubernetes.io/infra
- priority: 20
- profile: openshift-control-plane
- priority: 30
- profile: openshift-node

The CR above is translated for the containerized TuneD daemon into its **recommend.conf** file based on the profile priorities. The profile with the highest priority (**10**) is **openshift-control-plane-es** and, therefore, it is considered first. The containerized TuneD daemon running on a given node looks to see if there is a pod running on the same node with the **tuned.openshift.io/elasticsearch** label set. If not, the entire **<match>** section evaluates as **false**. If there is such a pod with the label, in order for the **<match>** section to evaluate to **true**, the node label also needs to be **node-role.kubernetes.io/master** or **node-role.kubernetes.io/infra**.

If the labels for the profile with priority **10** matched, **openshift-control-plane-es** profile is applied and no other profile is considered. If the node/pod label combination did not match, the second highest priority profile (**openshift-control-plane**) is considered. This profile is applied if the containerized TuneD pod runs on a node with labels **node-role.kubernetes.io/master** or **node-role.kubernetes.io/infra**.

Finally, the profile **openshift-node** has the lowest priority of **30**. It lacks the **<match>** section and, therefore, will always match. It acts as a profile catch-all to set **openshift-node** profile, if no other profile with higher priority matches on a given node.



OPENSHIFT_10_0319

Example: Machine config pool based matching

```

apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: openshift-node-custom
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
    - data: |
        [main]
        summary=Custom OpenShift node profile with an additional kernel parameter
        include=openshift-node
        [bootloader]
        cmdline_openshift_node_custom=+skew_tick=1
      name: openshift-node-custom

  recommend:
    - machineConfigLabels:
        machineconfiguration.openshift.io/role: "worker-custom"
      priority: 20
      profile: openshift-node-custom

```

To minimize node reboots, label the target nodes with a label the machine config pool's node selector will match, then create the Tuned CR above and finally create the custom machine config pool itself.

Cloud provider-specific TuneD profiles

With this functionality, all Cloud provider-specific nodes can conveniently be assigned a TuneD profile specifically tailored to a given Cloud provider on a OpenShift Container Platform cluster. This can be accomplished without adding additional node labels or grouping nodes into machine config pools.

This functionality takes advantage of **spec.providerID** node object values in the form of **<cloud-provider>://<cloud-provider-specific-id>** and writes the file **/var/lib/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.

6.7.3. 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 ^ {} \;
```

6.7.4. 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](#)

6.8. REMEDIATING, FENCING, AND MAINTAINING NODES

When node-level failures occur, such as the kernel hangs or network interface controllers (NICs) fail, the work required from the cluster does not decrease, and workloads from affected nodes need to be restarted somewhere. Failures affecting these workloads risk data loss, corruption, or both. It is important to isolate the node, known as **fencing**, before initiating recovery of the workload, known as **remediation**, and recovery of the node.

For more information on remediation, fencing, and maintaining nodes, see the [Workload Availability for Red Hat OpenShift](#) documentation.

6.9. UNDERSTANDING NODE REBOOTING

To reboot a node without causing an outage for applications running on the platform, it is important to first evacuate the pods. For pods that are made highly available by the routing tier, nothing else needs to be done. For other pods needing storage, typically databases, it is critical to ensure that they can remain in operation with one pod temporarily going offline. While implementing resiliency for stateful pods is different for each application, in all cases it is important to configure the scheduler to use node anti-affinity to ensure that the pods are properly spread across available nodes.

Another challenge is how to handle nodes that are running critical infrastructure such as the router or the registry. The same node evacuation process applies, though it is important to understand certain edge cases.

6.9.1. About rebooting nodes running critical infrastructure

When rebooting nodes that host critical OpenShift Container Platform infrastructure components, such as router pods, registry pods, and monitoring pods, ensure that there are at least three nodes available to run these components.

The following scenario demonstrates how service interruptions can occur with applications running on OpenShift Container Platform when only two nodes are available:

- Node A is marked unschedulable and all pods are evacuated.
- The registry pod running on that node is now redeployed on node B. Node B is now running both registry pods.
- Node B is now marked unschedulable and is evacuated.
- The service exposing the two pod endpoints on node B loses all endpoints, for a brief period of time, until they are redeployed to node A.

When using three nodes for infrastructure components, this process does not result in a service disruption. However, due to pod scheduling, the last node that is evacuated and brought back into rotation does not have a registry pod. One of the other nodes has two registry pods. To schedule the third registry pod on the last node, use pod anti-affinity to prevent the scheduler from locating two registry pods on the same node.

Additional information

- For more information on pod anti-affinity, see [Placing pods relative to other pods using affinity and anti-affinity rules](#).

6.9.2. Rebooting a node using pod anti-affinity

Pod anti-affinity is slightly different than node anti-affinity. Node anti-affinity can be violated if there are no other suitable locations to deploy a pod. Pod anti-affinity can be set to either required or preferred.

With this in place, if only two infrastructure nodes are available and one is rebooted, the container image registry pod is prevented from running on the other node. **oc get pods** reports the pod as unready until a suitable node is available. Once a node is available and all pods are back in ready state, the next node can be restarted.

Procedure

To reboot a node using pod anti-affinity:

1. Edit the node specification to configure pod anti-affinity:

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-antiaffinity
spec:
```

```

affinity:
  podAntiAffinity: ❶
    preferredDuringSchedulingIgnoredDuringExecution: ❷
      - weight: 100 ❸
        podAffinityTerm:
          labelSelector:
            matchExpressions:
              - key: registry ❹
                operator: In ❺
                values:
                  - default
          topologyKey: kubernetes.io/hostname
#...

```

- ❶ Stanza to configure pod anti-affinity.
- ❷ Defines a preferred rule.
- ❸ Specifies a weight for a preferred rule. The node with the highest weight is preferred.
- ❹ Description of the pod label that determines when the anti-affinity rule applies. Specify a key and value for the label.
- ❺ The operator represents the relationship between the label on the existing pod and the set of values in the **matchExpression** parameters in the specification for the new pod. Can be **In**, **NotIn**, **Exists**, or **DoesNotExist**.

This example assumes the container image registry pod has a label of **registry=default**. Pod anti-affinity can use any Kubernetes match expression.

2. Enable the **MatchInterPodAffinity** scheduler predicate in the scheduling policy file.
3. Perform a graceful restart of the node.

6.9.3. Understanding how to reboot nodes running routers

In most cases, a pod running an OpenShift Container Platform router exposes a host port.

The **PodFitsPorts** scheduler predicate ensures that no router pods using the same port can run on the same node, and pod anti-affinity is achieved. If the routers are relying on IP failover for high availability, there is nothing else that is needed.

For router pods relying on an external service such as AWS Elastic Load Balancing for high availability, it is that service's responsibility to react to router pod restarts.

In rare cases, a router pod may not have a host port configured. In those cases, it is important to follow the recommended restart process for infrastructure nodes.

6.9.4. Rebooting a node gracefully

Before rebooting a node, it is recommended to backup etcd data to avoid any data loss on the node.



NOTE

For single-node OpenShift clusters that require users to perform the **oc login** command rather than having the certificates in **kubeconfig** file to manage the cluster, the **oc adm** commands might not be available after cordoning and draining the node. This is because the **openshift-oauth-apiserver** pod is not running due to the cordon. You can use SSH to access the nodes as indicated in the following procedure.

In a single-node OpenShift cluster, pods cannot be rescheduled when cordoning and draining. However, doing so gives the pods, especially your workload pods, time to properly stop and release associated resources.

Procedure

To perform a graceful restart of a node:

1. Mark the node as unschedulable:

```
$ oc adm cordon <node1>
```

2. Drain the node to remove all the running pods:

```
$ oc adm drain <node1> --ignore-daemonsets --delete-emptydir-data --force
```

You might receive errors that pods associated with custom pod disruption budgets (PDB) cannot be evicted.

Example error

```
error when evicting pods/"rails-postgresql-example-1-72v2w" -n "rails" (will retry after 5s):
Cannot evict pod as it would violate the pod's disruption budget.
```

In this case, run the drain command again, adding the **disable-eviction** flag, which bypasses the PDB checks:

```
$ oc adm drain <node1> --ignore-daemonsets --delete-emptydir-data --force --disable-
eviction
```

3. Access the node in debug mode:

```
$ oc debug node/<node1>
```

4. Change your root directory to **/host**:

```
$ chroot /host
```

5. Restart the node:

```
$ systemctl reboot
```

In a moment, the node enters the **NotReady** state.

**NOTE**

With some single-node OpenShift clusters, the **oc** commands might not be available after you cordon and drain the node because the **openshift-oauth-apiserver** pod is not running. You can use SSH to connect to the node and perform the reboot.

```
$ ssh core@<master-node>.<cluster_name>.<base_domain>
```

```
$ sudo systemctl reboot
```

6. After the reboot is complete, mark the node as schedulable by running the following command:

```
$ oc adm uncordon <node1>
```

**NOTE**

With some single-node OpenShift clusters, the **oc** commands might not be available after you cordon and drain the node because the **openshift-oauth-apiserver** pod is not running. You can use SSH to connect to the node and uncordon it.

```
$ ssh core@<target_node>
```

```
$ sudo oc adm uncordon <node> --kubeconfig /etc/kubernetes/static-pod-resources/kube-apiserver-certs/secrets/node-kubeconfigs/localhost.kubeconfig
```

7. Verify that the node is ready:

```
$ oc get node <node1>
```

Example output

```
NAME      STATUS  ROLES  AGE   VERSION
<node1> Ready worker 6d22h v1.18.3+b0068a8
```

Additional information

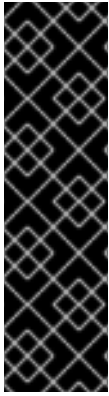
For information on etcd data backup, see [Backing up etcd data](#).

6.10. BOOT IMAGE MANAGEMENT

For Google Cloud and Amazon Web Services (AWS) clusters, by default the Machine Config Operator (MCO) manages and updates the boot image that is used to scale up your nodes. This means that by default, the MCO updates the boot images whenever you upgrade your cluster.

For VMware vSphere, you can enable boot image management as a Technology Preview feature.

For all other platforms, the MCO does not update the boot image with each cluster update.



IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

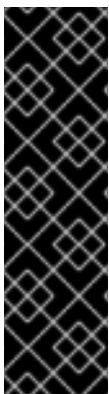
Additional resources

- [Enabling features using feature gates](#)

6.10.1. About boot image management

By default, for Google Cloud and Amazon Web Services (AWS) clusters, the Machine Config Operator (MCO) updates the boot image in the machine sets in your cluster whenever you update your cluster.

For VMware vSphere, you can enable boot image management as a Technology Preview feature. For information on how to enable this feature, see "Enabling boot image management".



IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

You can disable the boot image management feature, if needed. When the feature is disabled, the boot image no longer updates with the cluster. For example, with the feature disabled, if your cluster was originally created with OpenShift Container Platform 4.16, the boot image that the MCO would use to create nodes is the same 4.16 version, even if your cluster is at a later version.

However, using an older boot image could cause the following issues:

- Extra time to start nodes
- Certificate expiration issues
- Version skew issues

For information on how to disable this feature, see "Disabling boot image management". If you disable this feature, you can re-enable the feature at any time. For information, see "Enabling boot image management".

**NOTE**

The ability to configure boot image management is available for only Google Cloud and AWS clusters. It is not supported for clusters managed by the Cluster CAPI Operator.

How the cluster behaves after disabling or re-enabling the feature, depends upon when you made the change, including the following scenarios:

- If you disable the feature before updating to a new OpenShift Container Platform version:
 - The boot image version used by the machine sets remains the same OpenShift Container Platform version as when the feature was disabled.
 - When you scale up nodes, the new nodes use that same OpenShift Container Platform version.
- If you disable the feature after updating to a new OpenShift Container Platform version:
 - The boot image version used by the machine sets is updated to match the updated OpenShift Container Platform version.
 - When you scale up nodes, the new nodes use the updated OpenShift Container Platform version.
 - If you update to a later OpenShift Container Platform version, the boot image version in the machine sets remains at the current version and is not updated with the cluster.
- If you enable the feature after disabling:
 - The boot image version used by the machine sets is updated to the current OpenShift Container Platform version, if different.
 - When you scale up nodes, the new nodes use the current OpenShift Container Platform version in the cluster.

**NOTE**

Because a boot image is used only when a node is scaled up, this feature has no effect on existing nodes.

To view the current boot image used in your cluster, use one of the following methods, based on your platform:

- For Google Cloud and AWS, you can examine a machine set.

**NOTE**

The location and format of the boot image within the machine set differs, based on the platform. However, the boot image is always listed in the **spec.template.spec.providerSpec** parameter.

Example Google Cloud machine set with the boot image reference

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
```

```

metadata:
  name: ci-ln-hmy310k-72292-5f87z-worker-a
  namespace: openshift-machine-api
spec:
  # ...
  template:
    # ...
    spec:
      # ...
      providerSpec:
        # ...
        value:
          disks:
            - autoDelete: true
              boot: true
              image: projects/rhcos-cloud/global/images/rhcos-412-85-202203181601-0-gcp-x86-
64 1
    # ...

```

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

Example AWS machine set with the boot image reference

```

apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: ci-ln-hmy310k-72292-5f87z-worker-a
  namespace: openshift-machine-api
spec:
  # ...
  template:
    # ...
    spec:
      # ...
      providerSpec:
        value:
          ami:
            id: ami-0e8fd9094e487d1ff
    # ...

```

- For VMware vSphere, examine an affected node by opening an **oc debug** session to the node and using the **rpm-ostree status** command.

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

Example AWS node with the boot image reference

```

State: idle
Deployments:

```

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

IMPORTANT

If any of the machine sets for which you want to enable boot image management use a ***-user-data** secret that is based on Ignition version 2.2.0, the Machine Config Operator converts the Ignition version to 3.4.0 when you enable the feature. OpenShift Container Platform versions 4.5 and lower use Ignition version 2.2.0. If this conversion fails, the MCO or your cluster could degrade. An error message that includes *err: converting ignition stub failed: failed to parse Ignition config* is added to the output of the **oc get ClusterOperator machine-config** command. You can use the following general steps to correct the problem:

1. Disable the boot image management feature. For information, see "Disabling boot image management".
2. Manually update the ***-user-data** secret to use Ignition version to 3.2.0.
3. Enable the boot image management feature. For information, see "Enabling boot image management".

Additional resources

- [Disabling boot image management](#)
- [Enabling boot image management](#)

6.10.2. Disabling boot image management

By default, for Google Cloud and Amazon Web Services (AWS) clusters, the Machine Config Operator (MCO) manages and updates the boot image in the machine sets in your cluster whenever you update your cluster. For VMware vSphere, you can enable boot image management as a Technology Preview feature.

You can disable the boot image management feature for your cluster by editing the **MachineConfiguration** object. When disabled, the Machine Config Operator (MCO) no longer manages the boot image in your cluster and no longer updates the boot image with each cluster update.

Disabling this feature does not rollback the nodes or machine sets to the originally-installed boot image. The machine sets retain the boot image version that was present when the feature was disabled and is not updated if the cluster is upgraded to a new OpenShift Container Platform version in the future. This feature has no effect on existing nodes.

After disabling the feature, you can re-enable the feature at any time. For more information, see "Enabling updated boot images".

Procedure

1. Edit the **MachineConfiguration** object to disable the boot image management feature for some or all of your machine sets:

```
$ oc edit MachineConfiguration cluster
```

- Optional: Disable the feature for all machine sets:

```
apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  name: cluster
  namespace: openshift-machine-config-operator
spec:
  # ...
  managedBootImages: ❶
  machineManagers:
    - apiGroup: machine.openshift.io ❷
      resource: machinesets ❸
      selection:
        mode: None ❹
```

- ❶ Configures the boot image management feature.
- ❷ Specifies an API group. This must be **machine.openshift.io**.
- ❸ Specifies the resource within the specified API group to apply the change. This must be **machinesets**.
- ❹ Specifies that the feature is disabled for all machine sets in the cluster.

Verification

- When the affected nodes return to the **READY** state, view the current state of the boot image management feature by viewing the machine configuration object:

```
$ oc get machineconfiguration cluster -o yaml
```

Example machine set with the boot image reference

```
kind: MachineConfiguration
metadata:
  name: cluster
  # ...
status:
  conditions:
    - lastTransitionTime: "2025-05-01T20:11:49Z"
      message: Reconciled 2 of 4 MAPI MachineSets | Reconciled 0 of 0 CAPI MachineSets
        | Reconciled 0 of 0 CAPI MachineDeployments
      reason: BootImageUpdateConfigurationUpdated
      status: "True"
      type: BootImageUpdateProgressing
    - lastTransitionTime: "2025-05-01T19:30:13Z"
      message: 0 Degraded MAPI MachineSets | 0 Degraded CAPI MachineSets | 0 CAPI
MachineDeployments
      reason: BootImageUpdateConfigurationUpdated
      status: "False"
      type: BootImageUpdateDegraded
  managedBootImagesStatus:
```

```
machineManagers:
- apiGroup: machine.openshift.io
  resource: machinesets
  selection:
    mode: All
```

- When the affected nodes return to the **READY** state, check the current boot image by using one of the following methods:
 - For Google Cloud and AWS, get the boot image version by running the following command. The location and format of the boot image within the machine set differs, based on the platform. However, the boot image is always listed in the **spec.template.spec.providerSpec.** parameter.

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

Example machine set with the boot image reference

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: ci-ln-77hmkpt-72292-d4pxp
    update-boot-image: "true"
  name: ci-ln-77hmkpt-72292-d4pxp-worker-a
  namespace: openshift-machine-api
spec:
  # ...
  template:
    # ...
    spec:
      # ...
      providerSpec:
        # ...
        value:
          disks:
            - autoDelete: true
              boot: true
              image: projects/rhcos-cloud/global/images/<boot_image> 1
        # ...
```

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

- For VMware vSphere, get the boot image version from an affected node:
 - a. Open an **oc debug** session to the node by running a command similar to the following:

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

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

```
sh-5.1# chroot /host
```

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

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

Example output

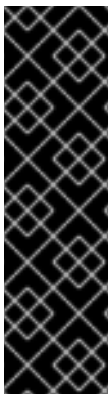
```
State: idle
Deployments:
* ostree-unverified-registry:quay.io/my-registry/...
  Digest: sha256:...
```

6.10.3. Enabling boot image management

By default, for Google Cloud and Amazon Web Services (AWS) clusters, the Machine Config Operator (MCO) updates the boot image in the machine sets in your cluster whenever you update your cluster.

If you disabled the boot image management feature, so that the boot images are not updated, you can re-enable the feature by editing the **MachineConfiguration** object.

For VMware vSphere, you can enable boot image management as a Technology Preview feature.



IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

Enabling the feature updates the boot image to the current OpenShift Container Platform version. If the cluster is again updated to a new OpenShift Container Platform version in the future, the boot image is updated again. New nodes created after enabling the feature use the updated boot image. This feature has no effect on existing nodes.

Prerequisites

- For vSphere, enable the **TechPreviewNoUpgrade** feature set on the cluster. For more information, see "Enabling features using feature gates".



NOTE

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

Wait until the **managedBootImagesStatus** stanza displays in the **MachineConfiguration** object.

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



```

kind: MachineConfiguration
metadata:
  name: cluster
# ...
status:
# ...
managedBootImagesStatus:
  machineManagers:
  - apiGroup: machine.openshift.io
    resource: machinesets
    selection:
      mode: None

```

Procedure

1. Edit the **MachineConfiguration** object, named **cluster**, to enable the boot image management feature for some or all of your machine sets:

```
$ oc edit MachineConfiguration cluster
```

- Optional: Enable the boot image management feature for all machine sets:

```

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

```

- ❶ Configures the boot image management feature.
- ❷ Specifies the API group. This must be **machine.openshift.io**.
- ❸ Specifies the resource within the specified API group to apply the change. This must be **machinesets**.
- ❹ Specifies that the feature is enabled for all machine sets in the cluster.

- Optional: Enable the boot image management feature for specific machine sets:

```

apiVersion: operator.openshift.io/v1
kind: MachineConfiguration
metadata:
  name: cluster
  namespace: openshift-machine-config-operator
spec:
# ...

```

```

managedBootImages: ❶
machineManagers:
- apiGroup: machine.openshift.io ❷
  resource: machinesets ❸
  selection:
    mode: Partial ❹
    partial:
      machineResourceSelector:
        matchLabels:
          region: "east"

```

- ❶ Configures the boot image update feature.
- ❷ Specifies the API group. This must be **machine.openshift.io**.
- ❸ Specifies the resource within the specified API group to apply the change. This must be **machinesets**.
- ❹ Specifies that the feature is enabled for machine sets with the specified label.

TIP

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

```
$ oc label machineset.machine ci-ln-hmy310k-72292-5f87z-worker-a region="east" -n openshift-machine-api
```

Verification

- When the affected nodes return to the **READY** state, view the current state of the boot image management feature by viewing the machine configuration object:

```
$ oc get machineconfiguration cluster -o yaml
```

Example machine set with the boot image reference

```

kind: MachineConfiguration
metadata:
  name: cluster
# ...
status:
  conditions:
  - lastTransitionTime: "2025-05-01T20:11:49Z"
    message: Reconciled 2 of 4 MAPI MachineSets | Reconciled 0 of 0 CAPI MachineSets
      | Reconciled 0 of 0 CAPI MachineDeployments
    reason: BootImageUpdateConfigurationUpdated
    status: "True"
    type: BootImageUpdateProgressing
  - lastTransitionTime: "2025-05-01T19:30:13Z"
    message: 0 Degraded MAPI MachineSets | 0 Degraded CAPI MachineSets | 0 CAPI
MachineDeployments
    reason: BootImageUpdateConfigurationUpdated

```

```

status: "False"
type: BootImageUpdateDegraded
managedBootImagesStatus:
  machineManagers:
    - apiGroup: machine.openshift.io
      resource: machinesets
      selection:
        mode: All

```

- When the affected nodes return to the **READY** state, check the current boot image by using one of the following methods:
 - For Google Cloud and AWS, get the boot image version by running the following command. The location and format of the boot image within the machine set differs, based on the platform. However, the boot image is always listed in the **spec.template.spec.providerSpec.** parameter.

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

Example machine set with the boot image reference

```

apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  labels:
    machine.openshift.io/cluster-api-cluster: ci-ln-77hmkpt-72292-d4pxp
    update-boot-image: "true"
  name: ci-ln-77hmkpt-72292-d4pxp-worker-a
  namespace: openshift-machine-api
spec:
  # ...
  template:
    # ...
    spec:
      # ...
      providerSpec:
        # ...
        value:
          disks:
            - autoDelete: true
              boot: true
              image: projects/rhcos-cloud/global/images/<boot_image> 1
          # ...

```

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

- For VMware vSphere, get the boot image version from an affected node:
 - a. Open an **oc debug** session to the node by running a command similar to the following:

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

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

```
sh-5.1# chroot /host
```

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

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

Example output

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

6.11. FREEING NODE RESOURCES USING GARBAGE COLLECTION

As an administrator, you can use OpenShift Container Platform to ensure that your nodes are running efficiently by freeing up resources through garbage collection.

The OpenShift Container Platform node performs two types of garbage collection:

- Container garbage collection: Removes terminated containers.
- Image garbage collection: Removes images not referenced by any running pods.

6.11.1. 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 6.5. 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.

6.11.2. 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 6.6. 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.

6.11.3. 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.

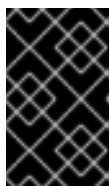
TIP

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 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

6.12. ALLOCATING RESOURCES FOR NODES IN AN OPENSIFT CONTAINER PLATFORM CLUSTER

To provide more reliable scheduling and minimize node resource overcommitment, reserve a portion of the CPU and memory resources for use by the underlying node components, such as **kubelet** and **kube-proxy**, and the remaining system components, such as **sshd** and **NetworkManager**. By specifying the resources to reserve, you provide the scheduler with more information about the remaining CPU and memory resources that a node has available for use by pods. You can allow OpenShift Container Platform to [automatically determine the optimal system-reserved CPU and memory resources](#) for your nodes or you can [manually determine and set the best resources](#) for your nodes.



IMPORTANT

To manually set resource values, you must use a kubelet config CR. You cannot use a machine config CR.

6.12.1. Understanding how to allocate resources for nodes

CPU and memory resources reserved for node components in OpenShift Container Platform are based on two node settings:

Setting	Description

Setting	Description
kube-reserved	This setting is not used with OpenShift Container Platform. Add the CPU and memory resources that you planned to reserve to the system-reserved setting.
system-reserved	This setting identifies the resources to reserve for the node components and system components, such as CRI-O and Kubelet. The default settings depend on the OpenShift Container Platform and Machine Config Operator versions. Confirm the default systemReserved parameter on the machine-config-operator repository.

If a flag is not set, the defaults are used. If none of the flags are set, the allocated resource is set to the node's capacity as it was before the introduction of allocatable resources.



NOTE

Any CPUs specifically reserved using the **reservedSystemCPUs** parameter are not available for allocation using **kube-reserved** or **system-reserved**.

6.12.1.1. How OpenShift Container Platform computes allocated resources

An allocated amount of a resource is computed based on the following formula:

$$[\text{Allocatable}] = [\text{Node Capacity}] - [\text{system-reserved}] - [\text{Hard-Eviction-Thresholds}]$$



NOTE

The withholding of **Hard-Eviction-Thresholds** from **Allocatable** improves system reliability because the value for **Allocatable** is enforced for pods at the node level.

If **Allocatable** is negative, it is set to **0**.

Each node reports the system resources that are used by the container runtime and kubelet. To simplify configuring the **system-reserved** parameter, view the resource use for the node by using the node summary API. The node summary is available at **/api/v1/nodes/<node>/proxy/stats/summary**.

6.12.1.2. How nodes enforce resource constraints

The node is able to limit the total amount of resources that pods can consume based on the configured allocatable value. This feature significantly improves the reliability of the node by preventing pods from using CPU and memory resources that are needed by system services such as the container runtime and node agent. To improve node reliability, administrators should reserve resources based on a target for resource use.

The node enforces resource constraints by using a new cgroup hierarchy that enforces quality of service. All pods are launched in a dedicated cgroup hierarchy that is separate from system daemons.

Administrators should treat system daemons similar to pods that have a guaranteed quality of service.

System daemons can burst within their bounding control groups and this behavior must be managed as part of cluster deployments. Reserve CPU and memory resources for system daemons by specifying the amount of CPU and memory resources in **system-reserved**.



NOTE

Enforcing **system-reserved** limits can prevent critical system services from receiving CPU and memory resources. As a result, a critical system service can be ended by the out-of-memory killer. The recommendation is to enforce **system-reserved** only if you have profiled the nodes exhaustively to determine precise estimates and you are confident that critical system services can recover if any process in that group is ended by the out-of-memory killer.

6.12.1.3. Understanding Eviction Thresholds

If a node is under memory pressure, it can impact the entire node and all pods running on the node. For example, a system daemon that uses more than its reserved amount of memory can trigger an out-of-memory event. To avoid or reduce the probability of system out-of-memory events, the node provides out-of-resource handling.

You can reserve some memory using the **--eviction-hard** flag. The node attempts to evict pods whenever memory availability on the node drops below the absolute value or percentage. If system daemons do not exist on a node, pods are limited to the memory **capacity - eviction-hard**. For this reason, resources set aside as a buffer for eviction before reaching out of memory conditions are not available for pods.

The following is an example to illustrate the impact of node allocatable for memory:

- Node capacity is **32Gi**
- **--system-reserved** is **3Gi**
- **--eviction-hard** is set to **100Mi**.

For this node, the effective node allocatable value is **28.9Gi**. If the node and system components use all their reservation, the memory available for pods is **28.9Gi**, and kubelet evicts pods when it exceeds this threshold.

If you enforce node allocatable, **28.9Gi**, with top-level cgroups, then pods can never exceed **28.9Gi**. Evictions are not performed unless system daemons consume more than **3.1Gi** of memory.

If system daemons do not use up all their reservation, with the above example, pods would face memcg OOM kills from their bounding cgroup before node evictions kick in. To better enforce QoS under this situation, the node applies the hard eviction thresholds to the top-level cgroup for all pods to be **Node Allocatable + Eviction Hard Thresholds**.

If system daemons do not use up all their reservation, the node will evict pods whenever they consume more than **28.9Gi** of memory. If eviction does not occur in time, a pod will be OOM killed if pods consume **29Gi** of memory.

6.12.1.4. How the scheduler determines resource availability

The scheduler uses the value of **node.Status.Allocatable** instead of **node.Status.Capacity** to decide if a node will become a candidate for pod scheduling.

By default, the node will report its machine capacity as fully schedulable by the cluster.

6.12.2. Understanding process ID limits

A process identifier (PID) is a unique identifier assigned by the Linux kernel to each process or thread currently running on a system. The number of processes that can run simultaneously on a system is limited to 4,194,304 by the Linux kernel. This number might also be affected by limited access to other system resources such as memory, CPU, and disk space.

In OpenShift Container Platform, consider these two supported limits for process ID (PID) usage before you schedule work on your cluster:

- Maximum number of PIDs per pod.
The default value is 4,096 in OpenShift Container Platform 4.11 and later. This value is controlled by the **podPidsLimit** parameter set on the node.

You can view the current PID limit on a node by running the following command in a **chroot** environment:

```
sh-5.1# cat /etc/kubernetes/kubelet.conf | grep -i pids
```

Example output

```
"podPidsLimit": 4096,
```

You can change the **podPidsLimit** by using a **KubeletConfig** object. See "Creating a KubeletConfig CR to edit kubelet parameters".

Containers inherit the **podPidsLimit** value from the parent pod, so the kernel enforces the lower of the two limits. For example, if the container PID limit is set to the maximum, but the pod PID limit is **4096**, the PID limit of each container in the pod is confined to 4096.

- Maximum number of PIDs per node.
The default value depends on node resources. In OpenShift Container Platform, this value is controlled by the **systemReserved** parameter in a kubelet configuration, which reserves PIDs on each node based on the total resources of the node. For more information, see "Allocating resources for nodes in an OpenShift Container Platform cluster".

When a pod exceeds the allowed maximum number of PIDs per pod, the pod might stop functioning correctly and might be evicted from the node. See [the Kubernetes documentation for eviction signals and thresholds](#) for more information.

When a node exceeds the allowed maximum number of PIDs per node, the node can become unstable because new processes cannot have PIDs assigned. If existing processes cannot complete without creating additional processes, the entire node can become unusable and require reboot. This situation can result in data loss, depending on the processes and applications being run. Customer administrators and Red Hat Site Reliability Engineering are notified when this threshold is reached, and a **Worker node is experiencing PIDPressure** warning will appear in the cluster logs.

Additional resources

- [Creating a KubeletConfig CR to edit kubelet parameters](#)
- [Allocating resources for nodes in an OpenShift Container Platform cluster](#)

6.12.2.1. Risks of setting higher process ID limits for OpenShift Container Platform pods

The **podPidsLimit** parameter for a pod controls the maximum number of processes and threads that can run simultaneously in that pod.

You can increase the value for **podPidsLimit** from the default of 4,096 to a maximum of 16,384. Changing this value might incur downtime for applications, because changing the **podPidsLimit** requires rebooting the affected node.

If you are running a large number of pods per node, and you have a high **podPidsLimit** value on your nodes, you risk exceeding the PID maximum for the node.

To find the maximum number of pods that you can run simultaneously on a single node without exceeding the PID maximum for the node, divide 3,650,000 by your **podPidsLimit** value. For example, if your **podPidsLimit** value is 16,384, and you expect the pods to use close to that number of process IDs, you can safely run 222 pods on a single node.



NOTE

Memory, CPU, and available storage can also limit the maximum number of pods that can run simultaneously, even when the **podPidsLimit** value is set appropriately.

6.12.3. Automatically allocating resources for nodes

OpenShift Container Platform can automatically determine the optimal **system-reserved** CPU and memory resources for nodes associated with a specific machine config pool and update the nodes with those values when the nodes start. By default, the **system-reserved** CPU is **500m** and **system-reserved** memory is **1Gi**.

To automatically determine and allocate the **system-reserved** resources on nodes, create a **KubeletConfig** custom resource (CR) to set the **autoSizingReserved: true** parameter. A script on each node calculates the optimal values for the respective reserved resources based on the installed CPU and memory capacity on each node. The script takes into account that increased capacity requires a corresponding increase in the reserved resources.

Automatically determining the optimal **system-reserved** settings ensures that your cluster is running efficiently and prevents node failure due to resource starvation of system components, such as CRI-O and kubelet, without your needing to manually calculate and update the values.

This feature is disabled by default.

Prerequisites

1. Obtain the label associated with the static **MachineConfigPool** object 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 an appropriate 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 resource allocation CR

```

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: dynamic-node 1
spec:
  autoSizingReserved: true 2
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" 3
#...
```

- 1 Assign a name to CR.
- 2 Add the **autoSizingReserved** parameter set to **true** to allow OpenShift Container Platform to automatically determine and allocate the **system-reserved** resources on the nodes associated with the specified label. To disable automatic allocation on those nodes, set this parameter to **false**.
- 3 Specify the label from the machine config pool that you configured in the "Prerequisites" section. You can choose any desired labels for the machine config pool, such as **custom-kubelet: small-pods**, or the default label, **pools.operator.machineconfiguration.openshift.io/worker: ""**.

The previous example enables automatic resource allocation on all worker nodes. OpenShift Container Platform drains the nodes, applies the kubelet config, and restarts the nodes.

2. Create the CR by entering the following command:

■

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

Verification

1. Log in to a node you configured by entering the following command:

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

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

```
# chroot /host
```

3. View the **/etc/node-sizing.env** file:

Example output

```
SYSTEM_RESERVED_MEMORY=3Gi
SYSTEM_RESERVED_CPU=0.08
```

The kubelet uses the **system-reserved** values in the **/etc/node-sizing.env** file. In the previous example, the worker nodes are allocated **0.08** CPU and 3 Gi of memory. It can take several minutes for the optimal values to appear.

6.12.4. Manually allocating resources for nodes

OpenShift Container Platform supports the CPU and memory resource types for allocation. The **ephemeral-resource** resource type is also supported. For the **cpu** type, you specify the resource quantity in units of cores, such as **200m**, **0.5**, or **1**. For **memory** and **ephemeral-storage**, you specify the resource quantity in units of bytes, such as **200Ki**, **50Mi**, or **5Gi**. By default, the **system-reserved** CPU is **500m** and **system-reserved** memory is **1Gi**.

As an administrator, you can set these values by using a kubelet config custom resource (CR) through a set of **<resource_type>=<resource_quantity>** pairs (e.g., **cpu=200m,memory=512Mi**).



IMPORTANT

You must use a kubelet config CR to manually set resource values. You cannot use a machine config CR.

For details on the recommended **system-reserved** values, refer to the [recommended system-reserved values](#).

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 resource allocation CR

```

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-allocatable 1
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" 2
  kubeletConfig:
    systemReserved: 3
    cpu: 1000m
    memory: 1Gi
#...
```

- 1 Assign a name to CR.
- 2 Specify the label from the machine config pool.
- 3 Specify the resources to reserve for the node components and system components.

2. Run the following command to create the CR:

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


6.13. ALLOCATING SPECIFIC CPUS FOR NODES IN A CLUSTER

When using the [static CPU Manager policy](#), you can reserve specific CPUs for use by specific nodes in your cluster. For example, on a system with 24 CPUs, you could reserve CPUs numbered 0 - 3 for the control plane allowing the compute nodes to use CPUs 4 - 23.

6.13.1. Reserving CPUs for nodes

To explicitly define a list of CPUs that are reserved for specific nodes, create a **KubeletConfig** custom resource (CR) to define the **reservedSystemCPUs** parameter. This list supersedes the CPUs that might be reserved using the **systemReserved** parameter.

For more information on the **systemReserved** parameter, see "Allocating resources for nodes in an OpenShift Container Platform cluster".

Procedure

1. Obtain the label associated with the machine config pool (MCP) for the type of node you want to configure:

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

For example:

```
$ oc describe machineconfigpool worker
```

Example output

```
Name:      worker
Namespace:
Labels:    machineconfiguration.openshift.io/mco-built-in=
           pools.operator.machineconfiguration.openshift.io/worker= 1
Annotations: <none>
API Version: machineconfiguration.openshift.io/v1
Kind:      MachineConfigPool
#...
```

- 1 Get the MCP label.

2. Create a YAML file for the **KubeletConfig** CR:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-reserved-cpus 1
spec:
  kubeletConfig:
    reservedSystemCPUs: "0,1,2,3" 2
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" 3
#...
```

- 1 Specify a name for the CR.
- 2 Specify the core IDs of the CPUs you want to reserve for the nodes associated with the MCP.
- 3 Specify the label from the MCP.

3. Create the CR object:

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

Additional resources

- For more information on the **systemReserved** parameter, see [Allocating resources for nodes in an OpenShift Container Platform cluster](#).

6.14. ENABLING TLS SECURITY PROFILES FOR THE KUBELET

You can use a TLS (Transport Layer Security) security profile to define which TLS ciphers are required by the kubelet when it is acting as an HTTP server. The kubelet uses its HTTP/GRPC server to communicate with the Kubernetes API server, which sends commands to pods, gathers logs, and run exec commands on pods through the kubelet.

A TLS security profile defines the TLS ciphers that the Kubernetes API server must use when connecting with the kubelet to protect communication between the kubelet and the Kubernetes API server.



NOTE

By default, when the kubelet acts as a client with the Kubernetes API server, it automatically negotiates the TLS parameters with the API server.

6.14.1. Understanding TLS security profiles

You can use a TLS (Transport Layer Security) security profile to define which TLS ciphers are required by various OpenShift Container Platform components. The OpenShift Container Platform TLS security profiles are based on [Mozilla recommended configurations](#).

You can specify one of the following TLS security profiles for each component:

Table 6.7. TLS security profiles

Profile	Description

Profile	Description
Old	This profile is intended for use with legacy clients or libraries. The profile is based on the Old backward compatibility recommended configuration. The Old profile requires a minimum TLS version of 1.0.  NOTE For the Ingress Controller, the minimum TLS version is converted from 1.0 to 1.1.
Intermediate	This profile is the default TLS security profile for the Ingress Controller, kubelet, and control plane. The profile is based on the Intermediate compatibility recommended configuration. The Intermediate profile requires a minimum TLS version of 1.2.  NOTE This profile is the recommended configuration for the majority of clients.
Modern	This profile is intended for use with modern clients that have no need for backwards compatibility. This profile is based on the Modern compatibility recommended configuration. The Modern profile requires a minimum TLS version of 1.3.
Custom	This profile allows you to define the TLS version and ciphers to use.  WARNING Use caution when using a Custom profile, because invalid configurations can cause problems.

**NOTE**

When using one of the predefined profile types, the effective profile configuration is subject to change between releases. For example, given a specification to use the Intermediate profile deployed on release X.Y.Z, an upgrade to release X.Y.Z+1 might cause a new profile configuration to be applied, resulting in a rollout.

6.14.2. Configuring the TLS security profile for the kubelet

To configure a TLS security profile for the kubelet when it is acting as an HTTP server, create a **KubeletConfig** custom resource (CR) to specify a predefined or custom TLS security profile for specific nodes. If a TLS security profile is not configured, the default TLS security profile is **Intermediate**.

Sample KubeletConfig CR that configures the Old TLS security profile on worker nodes

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
# ...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ""
# ...
```

You can see the ciphers and the minimum TLS version of the configured TLS security profile in the **kubelet.conf** file on a configured node.

Prerequisites

- You are logged in to OpenShift Container Platform as a user with the **cluster-admin** role.

Procedure

- Create a **KubeletConfig** CR to configure the TLS security profile:

Sample KubeletConfig CR for a Custom profile

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-kubelet-tls-security-profile
spec:
  tlsSecurityProfile:
    type: Custom 1
    custom: 2
      ciphers: 3
        - ECDHE-ECDSA-CHACHA20-POLY1305
        - ECDHE-RSA-CHACHA20-POLY1305
        - ECDHE-RSA-AES128-GCM-SHA256
        - ECDHE-ECDSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
    machineConfigPoolSelector:
      matchLabels:
        pools.operator.machineconfiguration.openshift.io/worker: "" 4
#...
```

- Specify the TLS security profile type (**Old**, **Intermediate**, or **Custom**). The default is **Intermediate**.

- 2 Specify the appropriate field for the selected type:
 - **old:** {}
 - **intermediate:** {}
 - **modern:** {}
 - **custom:**
- 3 For the **custom** type, specify a list of TLS ciphers and minimum accepted TLS version.
- 4 Optional: Specify the machine config pool label for the nodes you want to apply the TLS security profile.

2. Create the **KubeletConfig** object:

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

Depending on the number of worker nodes in the cluster, wait for the configured nodes to be rebooted one by one.

Verification

To verify that the profile is set, perform the following steps after the nodes are in the **Ready** state:

1. Start a debug session for a configured node:

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

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

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

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

```
sh-4.4# cat /etc/kubernetes/kubelet.conf
```

Example output

```
"kind": "KubeletConfiguration",
"apiVersion": "kubelet.config.k8s.io/v1beta1",
#...
"tlsCipherSuites": [
  "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256",
  "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256",
  "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384",
  "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384",
  "TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256",
  "TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256"
],
"tlsMinVersion": "VersionTLS12",
#...
```

6.15. CREATING INFRASTRUCTURE NODES



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}'
```

You can use infrastructure machine sets to create machines that host only infrastructure components, such as the default router, the integrated container image registry, and the 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.



NOTE

After adding the **NoSchedule** taint on the infrastructure node, existing DNS pods running on that node are marked as **misscheduled**. You must either delete or [add toleration on misscheduled DNS pods](#).

6.15.1. OpenShift Container Platform infrastructure components

Each self-managed Red Hat OpenShift subscription includes entitlements for OpenShift Container Platform and other OpenShift-related components. These entitlements are included for running OpenShift Container Platform control plane and infrastructure workloads and do not need to be accounted for during sizing.

To qualify as an infrastructure node and use the included entitlement, only components that are supporting the cluster, and not part of an end-user application, can run on those instances. Examples include the following components:

- Kubernetes and OpenShift Container Platform control plane services
- The default router
- The integrated container image registry
- The HAProxy-based Ingress Controller
- The cluster metrics collection, or monitoring service, including components for monitoring user-defined projects
- Cluster aggregated logging
- Red Hat Quay

- Red Hat OpenShift Data Foundation
- Red Hat Advanced Cluster Management for Kubernetes
- Red Hat Advanced Cluster Security for Kubernetes
- Red Hat OpenShift GitOps
- Red Hat OpenShift Pipelines
- Red Hat OpenShift Service Mesh

Any node that runs any other container, pod, or component is a worker node that your subscription must cover.

For information about infrastructure nodes and which components can run on infrastructure nodes, see the "Red Hat OpenShift control plane and infrastructure nodes" section in the [OpenShift sizing and subscription guide for enterprise Kubernetes](#) document.

To create an infrastructure node, you can [use a machine set](#), [label the node](#), or [use a machine config pool](#).

6.15.1.1. 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

- [Moving resources to infrastructure machine sets](#)

CHAPTER 7. WORKING WITH CONTAINERS

7.1. UNDERSTANDING CONTAINERS

The basic units of OpenShift Container Platform applications are called *containers*. [Linux container technologies](#) are lightweight mechanisms for isolating running processes so that they are limited to interacting with only their designated resources.

Many application instances can be running in containers on a single host without visibility into each others' processes, files, network, and so on. Typically, each container provides a single service (often called a "micro-service"), such as a web server or a database, though containers can be used for arbitrary workloads.

The Linux kernel has been incorporating capabilities for container technologies for years. OpenShift Container Platform and Kubernetes add the ability to orchestrate containers across multi-host installations.

7.1.1. About containers and RHEL kernel memory

Due to Red Hat Enterprise Linux (RHEL) behavior, a container on a node with high CPU usage might seem to consume more memory than expected. The higher memory consumption could be caused by the **kmem_cache** in the RHEL kernel. The RHEL kernel creates a **kmem_cache** for each cgroup. For added performance, the **kmem_cache** contains a **cpu_cache**, and a node cache for any NUMA nodes. These caches all consume kernel memory.

The amount of memory stored in those caches is proportional to the number of CPUs that the system uses. As a result, a higher number of CPUs results in a greater amount of kernel memory being held in these caches. Higher amounts of kernel memory in these caches can cause OpenShift Container Platform containers to exceed the configured memory limits, resulting in the container being killed.

To avoid losing containers due to kernel memory issues, ensure that the containers request sufficient memory. You can use the following formula to estimate the amount of memory consumed by the **kmem_cache**, where **nproc** is the number of processing units available that are reported by the **nproc** command. The lower limit of container requests should be this value plus the container memory requirements:

$$\$(nproc) \times 1/2 \text{ MiB}$$

7.1.2. About the container engine and container runtime

A *container engine* is a piece of software that processes user requests, including command-line options and image pulls. The container engine uses a *container runtime*, also called a *lower-level container runtime*, to run and manage the components required to deploy and operate containers. You likely will not need to interact with the container engine or container runtime.



NOTE

The OpenShift Container Platform documentation uses the term *container runtime* to refer to the lower-level container runtime. Other documentation can refer to the container engine as the container runtime.

OpenShift Container Platform uses CRI-O as the container engine and runC or crun as the container runtime. The default container runtime is crun. Both container runtimes adhere to the [Open Container Initiative \(OCI\)](#) runtime specifications.

CRI-O is a Kubernetes-native container engine implementation that integrates closely with the operating system to deliver an efficient and optimized Kubernetes experience. The CRI-O container engine runs as a systemd service on each OpenShift Container Platform cluster node.

crun, developed by Red Hat, is a fast and low-memory container runtime fully written in C. runC, developed by Docker and maintained by the Open Container Project, is a lightweight, portable container runtime written in Go.

crun has several improvements over runC, including:

- Smaller binary
- Quicker processing
- Lower memory footprint

runC has some benefits over crun, including:

- Most popular OCI container runtime.
- Longer tenure in production.
- Default container runtime of CRI-O.

You can move between the two container runtimes as needed.

For information on setting which container runtime to use, see [Creating a ContainerRuntimeConfig CR to edit CRI-O parameters](#).

7.2. USING INIT CONTAINERS TO PERFORM TASKS BEFORE A POD IS DEPLOYED

OpenShift Container Platform provides *init containers*, which are specialized containers that run before application containers and can contain utilities or setup scripts not present in an app image.

7.2.1. Understanding Init Containers

You can use an Init Container resource to perform tasks before the rest of a pod is deployed.

A pod can have Init Containers in addition to application containers. Init containers allow you to reorganize setup scripts and binding code.

An Init Container can:

- Contain and run utilities that are not desirable to include in the app Container image for security reasons.
- Contain utilities or custom code for setup that is not present in an app image. For example, there is no requirement to make an image FROM another image just to use a tool like sed, awk, python, or dig during setup.

- Use Linux namespaces so that they have different filesystem views from app containers, such as access to secrets that application containers are not able to access.

Each Init Container must complete successfully before the next one is started. So, Init Containers provide an easy way to block or delay the startup of app containers until some set of preconditions are met.

For example, the following are some ways you can use Init Containers:

- Wait for a service to be created with a shell command like:

```
for i in {1..100}; do sleep 1; if dig myservice; then exit 0; fi; done; exit 1
```

- Register this pod with a remote server from the downward API with a command like:

```
$ curl -X POST
http://$MANAGEMENT_SERVICE_HOST:$MANAGEMENT_SERVICE_PORT/register -d
'instance=$()&ip=$()'
```

- Wait for some time before starting the app Container with a command like **sleep 60**.
- Clone a git repository into a volume.
- Place values into a configuration file and run a template tool to dynamically generate a configuration file for the main app Container. For example, place the POD_IP value in a configuration and generate the main app configuration file using Jinja.

See the [Kubernetes documentation](#) for more information.

7.2.2. Creating Init Containers

The following example outlines a simple pod which has two Init Containers. The first waits for **myservice** and the second waits for **mydb**. After both containers complete, the pod begins.

Procedure

1. Create the pod for the Init Container:
 - a. Create a YAML file similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
labels:
  app: myapp
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: myapp-container
    image: registry.access.redhat.com/ubi9/ubi:latest
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
```

```

    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
    initContainers:
    - name: init-myservice
      image: registry.access.redhat.com/ubi9/ubi:latest
      command: ['sh', '-c', 'until getent hosts myservice; do echo waiting for myservice; sleep
2; done;']
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
    - name: init-mydb
      image: registry.access.redhat.com/ubi9/ubi:latest
      command: ['sh', '-c', 'until getent hosts mydb; do echo waiting for mydb; sleep 2;
done;']
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]

```

- b. Create the pod:

```
$ oc create -f myapp.yaml
```

- c. View the status of the pod:

```
$ oc get pods
```

Example output

NAME	READY	STATUS	RESTARTS	AGE
myapp-pod	0/1	Init:0/2	0	5s

The pod status, **Init:0/2**, indicates it is waiting for the two services.

2. Create the **myservice** service.

- a. Create a YAML file similar to the following:

```

kind: Service
apiVersion: v1
metadata:
  name: myservice
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376

```

- b. Create the pod:

```
$ oc create -f myservice.yaml
```

- c. View the status of the pod:

```
$ oc get pods
```

Example output

NAME	READY	STATUS	RESTARTS	AGE
myapp-pod	0/1	Init:1/2	0	5s

The pod status, **Init:1/2**, indicates it is waiting for one service, in this case the **mydb** service.

3. Create the **mydb** service:

- a. Create a YAML file similar to the following:

```
kind: Service
apiVersion: v1
metadata:
  name: mydb
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9377
```

- b. Create the pod:

```
$ oc create -f mydb.yaml
```

- c. View the status of the pod:

```
$ oc get pods
```

Example output

NAME	READY	STATUS	RESTARTS	AGE
myapp-pod	1/1	Running	0	2m

The pod status indicated that it is no longer waiting for the services and is running.

7.3. USING VOLUMES TO PERSIST CONTAINER DATA

Files in a container are ephemeral. As such, when a container crashes or stops, the data is lost. You can use *volumes* to persist the data used by the containers in a pod. A volume is directory, accessible to the Containers in a pod, where data is stored for the life of the pod.

7.3.1. Understanding volumes

Volumes are mounted file systems available to pods and their containers which may be backed by a number of host-local or network attached storage endpoints. Containers are not persistent by default; on restart, their contents are cleared.

To ensure that the file system on the volume contains no errors and, if errors are present, to repair them when possible, OpenShift Container Platform invokes the **fsck** utility prior to the **mount** utility. This occurs when either adding a volume or updating an existing volume.

The simplest volume type is **emptyDir**, which is a temporary directory on a single machine. Administrators may also allow you to request a persistent volume that is automatically attached to your pods.



NOTE

emptyDir volume storage may be restricted by a quota based on the pod's FSGroup, if the FSGroup parameter is enabled by your cluster administrator.

7.3.2. Working with volumes using the OpenShift Container Platform CLI

You can use the CLI command **oc set volume** to add and remove volumes and volume mounts for any object that has a pod template like replication controllers or deployment configs. You can also list volumes in pods or any object that has a pod template.

The **oc set volume** command uses the following general syntax:

```
$ oc set volume <object_selection> <operation> <mandatory_parameters> <options>
```

Object selection

Specify one of the following for the **object_selection** parameter in the **oc set volume** command:

Table 7.1. Object Selection

Syntax	Description	Example
<object_type> <name>	Selects <name> of type <object_type> .	deploymentConfig registry
<object_type>/<name>	Selects <name> of type <object_type> .	deploymentConfig/registry
<object_type>--selector=<object_label_selector>	Selects resources of type <object_type> that matched the given label selector.	deploymentConfig--selector="name=registry"
<object_type> --all	Selects all resources of type <object_type> .	deploymentConfig --all
-f or --filename=<file_name>	File name, directory, or URL to file to use to edit the resource.	-f registry-deployment-config.json

Operation

Specify **--add** or **--remove** for the **operation** parameter in the **oc set volume** command.

Mandatory parameters

Any mandatory parameters are specific to the selected operation and are discussed in later sections.

Options

Any options are specific to the selected operation and are discussed in later sections.

7.3.3. Listing volumes and volume mounts in a pod

You can list volumes and volume mounts in pods or pod templates:

Procedure

To list volumes:

```
$ oc set volume <object_type>/<name> [options]
```

List volume supported options:

Option	Description	Default
--name	Name of the volume.	
-c, --containers	Select containers by name. It can also take wildcard '*' that matches any character.	'*'

For example:

- To list all volumes for pod **p1**:

```
$ oc set volume pod/p1
```

- To list volume **v1** defined on all deployment configs:

```
$ oc set volume dc --all --name=v1
```

7.3.4. Adding volumes to a pod

You can add volumes and volume mounts to a pod.

Procedure

To add a volume, a volume mount, or both to pod templates:

```
$ oc set volume <object_type>/<name> --add [options]
```

Table 7.2. Supported Options for Adding Volumes

Option	Description	Default
--name	Name of the volume.	Automatically generated, if not specified.

Option	Description	Default
-t, --type	Name of the volume source. Supported values: emptyDir , hostPath , secret , configmap , persistentVolumeClaim or projected .	emptyDir
-c, --containers	Select containers by name. It can also take wildcard '*' that matches any character.	'*'
-m, --mount-path	Mount path inside the selected containers. Do not mount to the container root, /, or any path that is the same in the host and the container. This can corrupt your host system if the container is sufficiently privileged, such as the host /dev/pts files. It is safe to mount the host by using /host .	
--path	Host path. Mandatory parameter for --type=hostPath . Do not mount to the container root, /, or any path that is the same in the host and the container. This can corrupt your host system if the container is sufficiently privileged, such as the host /dev/pts files. It is safe to mount the host by using /host .	
--secret-name	Name of the secret. Mandatory parameter for --type=secret .	
--configmap-name	Name of the configmap. Mandatory parameter for --type=configmap .	
--claim-name	Name of the persistent volume claim. Mandatory parameter for --type=persistentVolumeClaim .	
--source	Details of volume source as a JSON string. Recommended if the desired volume source is not supported by --type .	

Option	Description	Default
-o, --output	Display the modified objects instead of updating them on the server. Supported values: json , yaml .	
--output-version	Output the modified objects with the given version.	api-version

For example:

- To add a new volume source **emptyDir** to the **registry DeploymentConfig** object:

```
$ oc set volume dc/registry --add
```

TIP

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

Example 7.1. Sample deployment config with an added volume

```
kind: DeploymentConfig
apiVersion: apps.openshift.io/v1
metadata:
  name: registry
  namespace: registry
spec:
  replicas: 3
  selector:
    app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      volumes: 1
      - name: volume-pppsw
        emptyDir: {}
      containers:
      - name: httpd
        image: >-
          image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
        ports:
        - containerPort: 8080
          protocol: TCP
```

- 1 Add the volume source **emptyDir**.

- To add volume **v1** with secret **secret1** for replication controller **r1** and mount inside the containers at **/data**:

```
$ oc set volume rc/r1 --add --name=v1 --type=secret --secret-name='secret1' --mount-path=/data
```

TIP

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

Example 7.2. Sample replication controller with added volume and secret

```
kind: ReplicationController
apiVersion: v1
metadata:
  name: example-1
  namespace: example
spec:
  replicas: 0
  selector:
    app: httpd
    deployment: example-1
    deploymentconfig: example
  template:
    metadata:
      creationTimestamp: null
    labels:
      app: httpd
      deployment: example-1
      deploymentconfig: example
    spec:
      volumes: 1
      - name: v1
        secret:
          secretName: secret1
          defaultMode: 420
      containers:
      - name: httpd
        image: >-
          image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
        volumeMounts: 2
      - name: v1
        mountPath: /data
```

1

Add the volume and secret.

2

Add the container mount path.

- To add existing persistent volume **v1** with claim name **pvc1** to deployment configuration **dc.json** on disk, mount the volume on container **c1** at **/data**, and update the **DeploymentConfig** object on the server:

```
$ oc set volume -f dc.json --add --name=v1 --type=persistentVolumeClaim \
--claim-name=pvc1 --mount-path=/data --containers=c1
```

TIP

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

Example 7.3. Sample deployment config with persistent volume added

```
kind: DeploymentConfig
apiVersion: apps.openshift.io/v1
metadata:
  name: example
  namespace: example
spec:
  replicas: 3
  selector:
    app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      volumes:
        - name: volume-pppsw
          emptyDir: {}
        - name: v1 1
          persistentVolumeClaim:
            claimName: pvc1
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts: 2
            - name: v1
              mountPath: /data
```

- 1** Add the persistent volume claim named `pvc1`.
- 2** Add the container mount path.

- To add a volume `v1` based on Git repository <https://github.com/namespace1/project1> with revision `5125c45f9f563` for all replication controllers:

```
$ oc set volume rc --all --add --name=v1 \
--source='{ "gitRepo": {
  "repository": "https://github.com/namespace1/project1",
  "revision": "5125c45f9f563"
}}'
```

7.3.5. Updating volumes and volume mounts in a pod

You can modify the volumes and volume mounts in a pod.

Procedure

Updating existing volumes using the **--overwrite** option:

```
$ oc set volume <object_type>/<name> --add --overwrite [options]
```

For example:

- To replace existing volume **v1** for replication controller **r1** with existing persistent volume claim **pvc1**:

```
$ oc set volume rc/r1 --add --overwrite --name=v1 --type=persistentVolumeClaim --claim-name=pvc1
```

TIP

You can alternatively apply the following YAML to replace the volume:

Example 7.4. Sample replication controller with persistent volume claim named `pvc1`

```
kind: ReplicationController
apiVersion: v1
metadata:
  name: example-1
  namespace: example
spec:
  replicas: 0
  selector:
    app: httpd
    deployment: example-1
    deploymentconfig: example
  template:
    metadata:
      labels:
        app: httpd
        deployment: example-1
        deploymentconfig: example
    spec:
      volumes:
        - name: v1 1
          persistentVolumeClaim:
            claimName: pvc1
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts:
            - name: v1
              mountPath: /data
```

1 Set persistent volume claim to **pvc1**.

- To change the **DeploymentConfig** object **d1** mount point to `/opt` for volume **v1**:

```
$ oc set volume dc/d1 --add --overwrite --name=v1 --mount-path=/opt
```

TIP

You can alternatively apply the following YAML to change the mount point:

Example 7.5. Sample deployment config with mount point set to `opt`.

```
kind: DeploymentConfig
apiVersion: apps.openshift.io/v1
metadata:
  name: example
  namespace: example
spec:
  replicas: 3
  selector:
    app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      volumes:
        - name: volume-pppsw
          emptyDir: {}
        - name: v2
          persistentVolumeClaim:
            claimName: pvc1
        - name: v1
          persistentVolumeClaim:
            claimName: pvc1
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts: 1
            - name: v1
              mountPath: /opt
```

1

Set the mount point to **`opt`**.

7.3.6. Removing volumes and volume mounts from a pod

You can remove a volume or volume mount from a pod.

Procedure

To remove a volume from pod templates:

```
$ oc set volume <object_type>/<name> --remove [options]
```

Table 7.3. Supported options for removing volumes

Option	Description	Default
--name	Name of the volume.	
-c, --containers	Select containers by name. It can also take wildcard '*' that matches any character.	'*'
--confirm	Indicate that you want to remove multiple volumes at once.	
-o, --output	Display the modified objects instead of updating them on the server. Supported values: json , yaml .	
--output-version	Output the modified objects with the given version.	api-version

For example:

- To remove a volume **v1** from the **DeploymentConfig** object **d1**:

```
$ oc set volume dc/d1 --remove --name=v1
```

- To unmount volume **v1** from container **c1** for the **DeploymentConfig** object **d1** and remove the volume **v1** if it is not referenced by any containers on **d1**:

```
$ oc set volume dc/d1 --remove --name=v1 --containers=c1
```

- To remove all volumes for replication controller **r1**:

```
$ oc set volume rc/r1 --remove --confirm
```

7.3.7. Configuring volumes for multiple uses in a pod

You can configure a volume to share one volume for multiple uses in a single pod using the **volumeMounts.subPath** property to specify a **subPath** value inside a volume instead of the volume's root.



NOTE

You cannot add a **subPath** parameter to an existing scheduled pod.

Procedure

1. To view the list of files in the volume, run the **oc rsh** command:

```
$ oc rsh <pod>
```

Example output

```
sh-4.2$ ls /path/to/volume/subpath/mount
example_file1 example_file2 example_file3
```

2. Specify the **subPath**:

Example Pod spec with subPath parameter

```
apiVersion: v1
kind: Pod
metadata:
  name: my-site
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: mysql
      image: mysql
      volumeMounts:
        - mountPath: /var/lib/mysql
          name: site-data
          subPath: mysql 1
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
    - name: php
      image: php
      volumeMounts:
        - mountPath: /var/www/html
          name: site-data
          subPath: html 2
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  volumes:
    - name: site-data
      persistentVolumeClaim:
        claimName: my-site-data
```

1 Databases are stored in the **mysql** folder.

2 HTML content is stored in the **html** folder.

7.4. MAPPING VOLUMES USING PROJECTED VOLUMES

A *projected volume* maps several existing volume sources into the same directory.

The following types of volume sources can be projected:

- Secrets
- Config Maps
- Downward API



NOTE

All sources are required to be in the same namespace as the pod.

7.4.1. Understanding projected volumes

Projected volumes can map any combination of these volume sources into a single directory, allowing the user to:

- automatically populate a single volume with the keys from multiple secrets, config maps, and with downward API information, so that I can synthesize a single directory with various sources of information;
- populate a single volume with the keys from multiple secrets, config maps, and with downward API information, explicitly specifying paths for each item, so that I can have full control over the contents of that volume.



IMPORTANT

When the **RunAsUser** permission is set in the security context of a Linux-based pod, the projected files have the correct permissions set, including container user ownership. However, when the Windows equivalent **RunAsUsername** permission is set in a Windows pod, the kubelet is unable to correctly set ownership on the files in the projected volume.

Therefore, the **RunAsUsername** permission set in the security context of a Windows pod is not honored for Windows projected volumes running in OpenShift Container Platform.

The following general scenarios show how you can use projected volumes.

Config map, secrets, Downward API.

Projected volumes allow you to deploy containers with configuration data that includes passwords. An application using these resources could be deploying Red Hat OpenStack Platform (RHOSP) on Kubernetes. The configuration data might have to be assembled differently depending on if the services are going to be used for production or for testing. If a pod is labeled with production or testing, the downward API selector **metadata.labels** can be used to produce the correct RHOSP configs.

Config map + secrets.

Projected volumes allow you to deploy containers involving configuration data and passwords. For example, you might execute a config map with some sensitive encrypted tasks that are decrypted using a vault password file.

ConfigMap + Downward API.

Projected volumes allow you to generate a config including the pod name (available via the **metadata.name** selector). This application can then pass the pod name along with requests to easily determine the source without using IP tracking.

Secrets + Downward API.

Projected volumes allow you to use a secret as a public key to encrypt the namespace of the pod (available via the **metadata.namespace** selector). This example allows the Operator to use the application to deliver the namespace information securely without using an encrypted transport.

7.4.1.1. Example Pod specs

The following are examples of **Pod** specs for creating projected volumes.

Pod with a secret, a Downward API, and a config map

```

apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: container-test
    image: busybox
    volumeMounts: ❶
    - name: all-in-one
      mountPath: "/projected-volume" ❷
      readOnly: true ❸
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
    volumes: ❹
    - name: all-in-one ❺
      projected:
        defaultMode: 0400 ❻
        sources:
        - secret:
            name: mysecret ❼
            items:
            - key: username
              path: my-group/my-username ❽
        - downwardAPI: ❾
            items:
            - path: "labels"
              fieldRef:
                fieldPath: metadata.labels
            - path: "cpu_limit"
              resourceFieldRef:
                containerName: container-test
                resource: limits.cpu
        - configMap: ❿
            name: myconfigmap
            items:

```

```
- key: config
  path: my-group/my-config
  mode: 0777 11
```

- 1** Add a **volumeMounts** section for each container that needs the secret.
- 2** Specify a path to an unused directory where the secret will appear.
- 3** Set **readOnly** to **true**.
- 4** Add a **volumes** block to list each projected volume source.
- 5** Specify any name for the volume.
- 6** Set the execute permission on the files.
- 7** Add a secret. Enter the name of the secret object. Each secret you want to use must be listed.
- 8** Specify the path to the secrets file under the **mountPath**. Here, the secrets file is in */projected-volume/my-group/my-username*.
- 9** Add a Downward API source.
- 10** Add a ConfigMap source.
- 11** Set the mode for the specific projection



NOTE

If there are multiple containers in the pod, each container needs a **volumeMounts** section, but only one **volumes** section is needed.

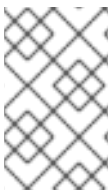
Pod with multiple secrets with a non-default permission mode set

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: container-test
      image: busybox
      volumeMounts:
        - name: all-in-one
          mountPath: "/projected-volume"
          readOnly: true
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

```

volumes:
- name: all-in-one
  projected:
    defaultMode: 0755
    sources:
    - secret:
        name: mysecret
        items:
        - key: username
          path: my-group/my-username
    - secret:
        name: mysecret2
        items:
        - key: password
          path: my-group/my-password
        mode: 511

```



NOTE

The **defaultMode** can only be specified at the projected level and not for each volume source. However, as illustrated above, you can explicitly set the **mode** for each individual projection.

7.4.1.2. Pathing Considerations

Collisions Between Keys when Configured Paths are Identical

If you configure any keys with the same path, the pod spec will not be accepted as valid. In the following example, the specified path for **mysecret** and **myconfigmap** are the same:

```

apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: container-test
    image: busybox
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  volumes:
  - name: all-in-one
    projected:
      sources:
      - secret:
          name: mysecret

```

```

items:
  - key: username
    path: my-group/data
- configMap:
  name: myconfigmap
  items:
    - key: config
      path: my-group/data

```

Consider the following situations related to the volume file paths.

Collisions Between Keys without Configured Paths

The only run-time validation that can occur is when all the paths are known at pod creation, similar to the above scenario. Otherwise, when a conflict occurs the most recent specified resource will overwrite anything preceding it (this is true for resources that are updated after pod creation as well).

Collisions when One Path is Explicit and the Other is Automatically Projected

In the event that there is a collision due to a user specified path matching data that is automatically projected, the latter resource will overwrite anything preceding it as before

7.4.2. Configuring a Projected Volume for a Pod

When creating projected volumes, consider the volume file path situations described in *Understanding projected volumes*.

The following example shows how to use a projected volume to mount an existing secret volume source. The steps can be used to create a user name and password secrets from local files. You then create a pod that runs one container, using a projected volume to mount the secrets into the same shared directory.

The user name and password values can be any valid string that is **base64** encoded.

The following example shows **admin** in base64:

```
$ echo -n "admin" | base64
```

Example output

```
YWRtaW4=
```

The following example shows the password **1f2d1e2e67df** in base64:

```
$ echo -n "1f2d1e2e67df" | base64
```

Example output

```
MWYyZDFIMmU2N2Rm
```

Procedure

To use a projected volume to mount an existing secret volume source.

1. Create the secret:

- a. Create a YAML file similar to the following, replacing the password and user information as appropriate:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  pass: MWYyZDFIMmU2N2Rm
  user: YWRtaW4=
```

- b. Use the following command to create the secret:

```
$ oc create -f <secrets-filename>
```

For example:

```
$ oc create -f secret.yaml
```

Example output

```
secret "mysecret" created
```

- c. You can check that the secret was created using the following commands:

```
$ oc get secret <secret-name>
```

For example:

```
$ oc get secret mysecret
```

Example output

```
NAME      TYPE      DATA   AGE
mysecret  Opaque    2       17h
```

```
$ oc get secret <secret-name> -o yaml
```

For example:

```
$ oc get secret mysecret -o yaml
```

```
apiVersion: v1
data:
  pass: MWYyZDFIMmU2N2Rm
  user: YWRtaW4=
kind: Secret
metadata:
  creationTimestamp: 2017-05-30T20:21:38Z
```

```

name: mysecret
namespace: default
resourceVersion: "2107"
selfLink: /api/v1/namespaces/default/secrets/mysecret
uid: 959e0424-4575-11e7-9f97-fa163e4bd54c
type: Opaque

```

2. Create a pod with a projected volume.

a. Create a YAML file similar to the following, including a **volumes** section:

```

kind: Pod
metadata:
  name: test-projected-volume
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: test-projected-volume
    image: busybox
    args:
    - sleep
    - "86400"
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
    securityContext:
      allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
  volumes:
  - name: all-in-one
    projected:
      sources:
      - secret:
          name: mysecret 1

```

1 The name of the secret you created.

b. Create the pod from the configuration file:

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

For example:

```
$ oc create -f secret-pod.yaml
```

Example output

```
pod "test-projected-volume" created
```

3. Verify that the pod container is running, and then watch for changes to the pod:

```
$ oc get pod <name>
```

For example:

```
$ oc get pod test-projected-volume
```

The output should appear similar to the following:

Example output

```
NAME                READY   STATUS    RESTARTS   AGE
test-projected-volume 1/1     Running   0           14s
```

4. In another terminal, use the **oc exec** command to open a shell to the running container:

```
$ oc exec -it <pod> <command>
```

For example:

```
$ oc exec -it test-projected-volume -- /bin/sh
```

5. In your shell, verify that the **projected-volumes** directory contains your projected sources:

```
/ # ls
```

Example output

```
bin      home      root      tmp
dev      proc      run       usr
etc      projected-volume sys       var
```

7.5. ALLOWING CONTAINERS TO CONSUME API OBJECTS

The *Downward API* is a mechanism that allows containers to consume information about API objects without coupling to OpenShift Container Platform. Such information includes the pod's name, namespace, and resource values. Containers can consume information from the downward API using environment variables or a volume plugin.

7.5.1. Expose pod information to Containers using the Downward API

The Downward API contains such information as the pod's name, project, and resource values. Containers can consume information from the downward API using environment variables or a volume plugin.

Fields within the pod are selected using the **FieldRef** API type. **FieldRef** has two fields:

Field	Description
fieldPath	The path of the field to select, relative to the pod.
apiVersion	The API version to interpret the fieldPath selector within.

Currently, the valid selectors in the v1 API include:

Selector	Description
metadata.name	The pod's name. This is supported in both environment variables and volumes.
metadata.namespace	The pod's namespace. This is supported in both environment variables and volumes.
metadata.labels	The pod's labels. This is only supported in volumes and not in environment variables.
metadata.annotations	The pod's annotations. This is only supported in volumes and not in environment variables.
status.podIP	The pod's IP. This is only supported in environment variables and not volumes.

The **apiVersion** field, if not specified, defaults to the API version of the enclosing pod template.

7.5.2. Understanding how to consume container values using the downward API

You containers can consume API values using environment variables or a volume plugin. Depending on the method you choose, containers can consume:

- Pod name
- Pod project/namespace
- Pod annotations
- Pod labels

Annotations and labels are available using only a volume plugin.

7.5.2.1. Consuming container values using environment variables

When using a container's environment variables, use the **EnvVar** type's **valueFrom** field (of type **EnvVarSource**) to specify that the variable's value should come from a **FieldRef** source instead of the literal value specified by the **value** field.

Only constant attributes of the pod can be consumed this way, as environment variables cannot be updated once a process is started in a way that allows the process to be notified that the value of a variable has changed. The fields supported using environment variables are:

- Pod name
- Pod project/namespace

Procedure

1. Create a new pod spec that contains the environment variables you want the container to consume:
 - a. Create a **pod.yaml** file similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: MY_POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
      restartPolicy: Never
# ...
```

- b. Create the pod from the **pod.yaml** file:

```
$ oc create -f pod.yaml
```

Verification

- Check the container's logs for the **MY_POD_NAME** and **MY_POD_NAMESPACE** values:

```
$ oc logs -p dapi-env-test-pod
```

7.5.2.2. Consuming container values using a volume plugin

You containers can consume API values using a volume plugin.

Containers can consume:

- Pod name
- Pod project/namespace
- Pod annotations
- Pod labels

Procedure

To use the volume plugin:

1. Create a new pod spec that contains the environment variables you want the container to consume:
 - a. Create a **volume-pod.yaml** file similar to the following:

```
kind: Pod
apiVersion: v1
metadata:
  labels:
    zone: us-east-coast
    cluster: downward-api-test-cluster1
    rack: rack-123
  name: dapi-volume-test-pod
  annotations:
    annotation1: "345"
    annotation2: "456"
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: volume-test-container
      image: gcr.io/google_containers/busybox
      command: ["sh", "-c", "cat /tmp/etc/pod_labels /tmp/etc/pod_annotations"]
      volumeMounts:
        - name: podinfo
          mountPath: /tmp/etc
          readOnly: false
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  volumes:
    - name: podinfo
      downwardAPI:
        defaultMode: 420
        items:
          - fieldRef:
```

```

        fieldPath: metadata.name
      path: pod_name
    - fieldRef:
        fieldPath: metadata.namespace
      path: pod_namespace
    - fieldRef:
        fieldPath: metadata.labels
      path: pod_labels
    - fieldRef:
        fieldPath: metadata.annotations
      path: pod_annotations
    restartPolicy: Never
  # ...

```

- b. Create the pod from the **volume-pod.yaml** file:

```
$ oc create -f volume-pod.yaml
```

Verification

- Check the container's logs and verify the presence of the configured fields:

```
$ oc logs -p dapi-volume-test-pod
```

Example output

```

cluster=downward-api-test-cluster1
rack=rack-123
zone=us-east-coast
annotation1=345
annotation2=456
kubernetes.io/config.source=api

```

7.5.3. Understanding how to consume container resources using the Downward API

When creating pods, you can use the Downward API to inject information about computing resource requests and limits so that image and application authors can correctly create an image for specific environments.

You can do this using environment variable or a volume plugin.

7.5.3.1. Consuming container resources using environment variables

When creating pods, you can use the Downward API to inject information about computing resource requests and limits using environment variables.

When creating the pod configuration, specify environment variables that correspond to the contents of the **resources** field in the **spec.container** field.



NOTE

If the resource limits are not included in the container configuration, the downward API defaults to the node's CPU and memory allocatable values.

Procedure

1. Create a new pod spec that contains the resources you want to inject:
 - a. Create a **pod.yaml** file similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
  - name: test-container
    image: gcr.io/google_containers/busybox:1.24
    command: [ "/bin/sh", "-c", "env" ]
    resources:
      requests:
        memory: "32Mi"
        cpu: "125m"
      limits:
        memory: "64Mi"
        cpu: "250m"
    env:
      - name: MY_CPU_REQUEST
        valueFrom:
          resourceFieldRef:
            resource: requests.cpu
      - name: MY_CPU_LIMIT
        valueFrom:
          resourceFieldRef:
            resource: limits.cpu
      - name: MY_MEM_REQUEST
        valueFrom:
          resourceFieldRef:
            resource: requests.memory
      - name: MY_MEM_LIMIT
        valueFrom:
          resourceFieldRef:
            resource: limits.memory
    # ...
```

- b. Create the pod from the **pod.yaml** file:

```
$ oc create -f pod.yaml
```

7.5.3.2. Consuming container resources using a volume plugin

When creating pods, you can use the Downward API to inject information about computing resource requests and limits using a volume plugin.

When creating the pod configuration, use the **spec.volumes.downwardAPI.items** field to describe the desired resources that correspond to the **spec.resources** field.



NOTE

If the resource limits are not included in the container configuration, the Downward API defaults to the node's CPU and memory allocatable values.

Procedure

1. Create a new pod spec that contains the resources you want to inject:
 - a. Create a **pod.yaml** file similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: client-container
      image: gcr.io/google_containers/busybox:1.24
      command: ["sh", "-c", "while true; do echo; if [[ -e /etc/cpu_limit ]]; then cat
/et /cpu_limit; fi; if [[ -e /etc/cpu_request ]]; then cat /etc/cpu_request; fi; if [[ -e
/et /mem_limit ]]; then cat /etc/mem_limit; fi; if [[ -e /etc/mem_request ]]; then cat
/et /mem_request; fi; sleep 5; done"]
      resources:
        requests:
          memory: "32Mi"
          cpu: "125m"
        limits:
          memory: "64Mi"
          cpu: "250m"
      volumeMounts:
        - name: podinfo
          mountPath: /etc
          readOnly: false
  volumes:
    - name: podinfo
      downwardAPI:
        items:
          - path: "cpu_limit"
            resourceFieldRef:
              containerName: client-container
              resource: limits.cpu
          - path: "cpu_request"
            resourceFieldRef:
              containerName: client-container
              resource: requests.cpu
          - path: "mem_limit"
            resourceFieldRef:
              containerName: client-container
              resource: limits.memory
          - path: "mem_request"
            resourceFieldRef:
              containerName: client-container
              resource: requests.memory
# ...
```

- b. Create the pod from the **volume-pod.yaml** file:

```
$ oc create -f volume-pod.yaml
```

7.5.4. Consuming secrets using the Downward API

When creating pods, you can use the downward API to inject secrets so image and application authors can create an image for specific environments.

Procedure

1. Create a secret to inject:
 - a. Create a **secret.yaml** file similar to the following:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
data:
  password: <password>
  username: <username>
type: kubernetes.io/basic-auth
```

- b. Create the secret object from the **secret.yaml** file:

```
$ oc create -f secret.yaml
```

2. Create a pod that references the **username** field from the above **Secret** object:
 - a. Create a **pod.yaml** file similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: mysecret
              key: username
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
```

```

    drop: [ALL]
    restartPolicy: Never
    # ...

```

- b. Create the pod from the **pod.yaml** file:

```
$ oc create -f pod.yaml
```

Verification

- Check the container's logs for the **MY_SECRET_USERNAME** value:

```
$ oc logs -p dapi-env-test-pod
```

7.5.5. Consuming configuration maps using the Downward API

When creating pods, you can use the Downward API to inject configuration map values so image and application authors can create an image for specific environments.

Procedure

1. Create a config map with the values to inject:
 - a. Create a **configmap.yaml** file similar to the following:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: myconfigmap
data:
  mykey: myvalue

```

- b. Create the config map from the **configmap.yaml** file:

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

2. Create a pod that references the above config map:
 - a. Create a **pod.yaml** file similar to the following:

```

apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]

```



```

env:
  - name: MY_CONFIGMAP_VALUE
    valueFrom:
      configMapKeyRef:
        name: myconfigmap
        key: mykey
securityContext:
  allowPrivilegeEscalation: false
capabilities:
  drop: [ALL]
restartPolicy: Always
# ...

```

- b. Create the pod from the **pod.yaml** file:

```
$ oc create -f pod.yaml
```

Verification

- Check the container's logs for the **MY_CONFIGMAP_VALUE** value:

```
$ oc logs -p dapi-env-test-pod
```

7.5.6. Referencing environment variables

When creating pods, you can reference the value of a previously defined environment variable by using the **\$()** syntax. If the environment variable reference can not be resolved, the value will be left as the provided string.

Procedure

1. Create a pod that references an existing environment variable:
 - a. Create a **pod.yaml** file similar to the following:

```

apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_EXISTING_ENV
          value: my_value
        - name: MY_ENV_VAR_REF_ENV
          value: $(MY_EXISTING_ENV)
      securityContext:

```

```

    allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
    restartPolicy: Never
  # ...

```

- b. Create the pod from the **pod.yaml** file:

```
$ oc create -f pod.yaml
```

Verification

- Check the container's logs for the **MY_ENV_VAR_REF_ENV** value:

```
$ oc logs -p dapi-env-test-pod
```

7.5.7. Escaping environment variable references

When creating a pod, you can escape an environment variable reference by using a double dollar sign. The value will then be set to a single dollar sign version of the provided value.

Procedure

1. Create a pod that references an existing environment variable:
 - a. Create a **pod.yaml** file similar to the following:

```

apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_NEW_ENV
          value: $$SOME_OTHER_ENV
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
      restartPolicy: Never
  # ...

```

- b. Create the pod from the **pod.yaml** file:

```
$ oc create -f pod.yaml
```

Verification

- Check the container's logs for the **MY_NEW_ENV** value:

```
$ oc logs -p dapi-env-test-pod
```

7.6. COPYING FILES TO OR FROM OPENSIFT CONTAINER PLATFORM CONTAINERS

You can use the CLI to copy local files to or from a remote directory in a container using the **rsync** command.

7.6.1. Understanding how to copy files

The **oc rsync** command, or remote sync, is a useful tool for copying database archives to and from your pods for backup and restore purposes. You can also use **oc rsync** to copy source code changes into a running pod for development debugging, when the running pod supports hot reload of source files.

```
$ oc rsync <source> <destination> [-c <container>]
```

7.6.1.1. Requirements

Specifying the Copy Source

The source argument of the **oc rsync** command must point to either a local directory or a pod directory. Individual files are not supported.

When specifying a pod directory the directory name must be prefixed with the pod name:

```
<pod name>:<dir>
```

If the directory name ends in a path separator (*/*), only the contents of the directory are copied to the destination. Otherwise, the directory and its contents are copied to the destination.

Specifying the Copy Destination

The destination argument of the **oc rsync** command must point to a directory. If the directory does not exist, but **rsync** is used for copy, the directory is created for you.

Deleting Files at the Destination

The **--delete** flag may be used to delete any files in the remote directory that are not in the local directory.

Continuous Syncing on File Change

Using the **--watch** option causes the command to monitor the source path for any file system changes, and synchronizes changes when they occur. With this argument, the command runs forever. Synchronization occurs after short quiet periods to ensure a rapidly changing file system does not result in continuous synchronization calls.

When using the **--watch** option, the behavior is effectively the same as manually invoking **oc rsync** repeatedly, including any arguments normally passed to **oc rsync**. Therefore, you can control the behavior via the same flags used with manual invocations of **oc rsync**, such as **--delete**.

7.6.2. Copying files to and from containers

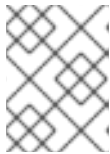
Support for copying local files to or from a container is built into the CLI.

Prerequisites

When working with **oc rsync**, note the following:

- rsync must be installed. The **oc rsync** command uses the local **rsync** tool, if present on the client machine and the remote container. If **rsync** is not found locally or in the remote container, a **tar** archive is created locally and sent to the container where the **tar** utility is used to extract the files. If **tar** is not available in the remote container, the copy will fail.

The **tar** copy method does not provide the same functionality as **oc rsync**. For example, **oc rsync** creates the destination directory if it does not exist and only sends files that are different between the source and the destination.



NOTE

In Windows, the **cwRsync** client should be installed and added to the PATH for use with the **oc rsync** command.

Procedure

- To copy a local directory to a pod directory:

```
$ oc rsync <local-dir> <pod-name>:/<remote-dir> -c <container-name>
```

For example:

```
$ oc rsync /home/user/source devpod1234:/src -c user-container
```

- To copy a pod directory to a local directory:

```
$ oc rsync devpod1234:/src /home/user/source
```

Example output

```
$ oc rsync devpod1234:/src/status.txt /home/user/
```

7.6.3. Using advanced Rsync features

The **oc rsync** command exposes fewer command-line options than standard **rsync**. In the case that you want to use a standard **rsync** command-line option that is not available in **oc rsync**, for example the **--exclude-from=FILE** option, it might be possible to use standard **rsync**'s **--rsh (-e)** option or **RSYNC_RSH** environment variable as a workaround, as follows:

```
$ rsync --rsh='oc rsh' --exclude-from=<file_name> <local-dir> <pod-name>:/<remote-dir>
```

or:

Export the **RSYNC_RSH** variable:

```
$ export RSYNC_RSH='oc rsh'
```

Then, run the `rsync` command:

```
$ rsync --exclude-from=<file_name> <local-dir> <pod-name>:/<remote-dir>
```

Both of the above examples configure standard **rsync** to use **oc rsh** as its remote shell program to enable it to connect to the remote pod, and are an alternative to running **oc rsync**.

7.7. EXECUTING REMOTE COMMANDS IN AN OPENSIFT CONTAINER PLATFORM CONTAINER

You can use the CLI to execute remote commands in OpenShift Container Platform containers.

7.7.1. Executing remote commands in containers

Support for remote container command execution is built into the CLI.

Procedure

To run a command in a container:

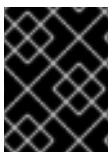
```
$ oc exec <pod> [-c <container>] -- <command> [<arg_1> ... <arg_n>]
```

For example:

```
$ oc exec mypod date
```

Example output

```
Thu Apr 9 02:21:53 UTC 2015
```



IMPORTANT

For security purposes, the **oc exec** command does not work when accessing privileged containers except when the command is executed by a **cluster-admin** user.

7.7.2. Protocol for initiating a remote command from a client

Clients initiate the execution of a remote command in a container by issuing a request to the Kubernetes API server:

```
/proxy/nodes/<node_name>/exec/<namespace>/<pod>/<container>?command=<command>
```

In the above URL:

- **<node_name>** is the FQDN of the node.
- **<namespace>** is the project of the target pod.
- **<pod>** is the name of the target pod.

- **<container>** is the name of the target container.
- **<command>** is the desired command to be executed.

For example:

```
/proxy/nodes/node123.openshift.com/exec/myns/mypod/mycontainer?command=date
```

Additionally, the client can add parameters to the request to indicate if:

- the client should send input to the remote container's command (stdin).
- the client's terminal is a TTY.
- the remote container's command should send output from stdout to the client.
- the remote container's command should send output from stderr to the client.

After sending an **exec** request to the API server, the client upgrades the connection to one that supports multiplexed streams; the current implementation uses **HTTP/2**.

The client creates one stream each for stdin, stdout, and stderr. To distinguish among the streams, the client sets the **streamType** header on the stream to one of **stdin**, **stdout**, or **stderr**.

The client closes all streams, the upgraded connection, and the underlying connection when it is finished with the remote command execution request.

7.8. USING PORT FORWARDING TO ACCESS APPLICATIONS IN A CONTAINER

OpenShift Container Platform supports port forwarding to pods.

7.8.1. Understanding port forwarding

You can use the CLI to forward one or more local ports to a pod. This allows you to listen on a given or random port locally, and have data forwarded to and from given ports in the pod.

Support for port forwarding is built into the CLI:

```
$ oc port-forward <pod> [<local_port>:]<remote_port> [...[<local_port_n>:]<remote_port_n>]
```

The CLI listens on each local port specified by the user, forwarding using the protocol described below.

Ports may be specified using the following formats:

5000	The client listens on port 5000 locally and forwards to 5000 in the pod.
6000:5000	The client listens on port 6000 locally and forwards to 5000 in the pod.
:5000 or 0:5000	The client selects a free local port and forwards to 5000 in the pod.

OpenShift Container Platform handles port-forward requests from clients. Upon receiving a request, OpenShift Container Platform upgrades the response and waits for the client to create port-forwarding streams. When OpenShift Container Platform receives a new stream, it copies data between the stream and the pod's port.

Architecturally, there are options for forwarding to a pod's port. The supported OpenShift Container Platform implementation invokes **nsenter** directly on the node host to enter the pod's network namespace, then invokes **socat** to copy data between the stream and the pod's port. However, a custom implementation could include running a *helper* pod that then runs **nsenter** and **socat**, so that those binaries are not required to be installed on the host.

7.8.2. Using port forwarding

You can use the CLI to port-forward one or more local ports to a pod.

Procedure

Use the following command to listen on the specified port in a pod:

```
$ oc port-forward <pod> [<local_port>:]<remote_port> [...[<local_port_n>:]<remote_port_n>]
```

For example:

- Use the following command to listen on ports **5000** and **6000** locally and forward data to and from ports **5000** and **6000** in the pod:

```
$ oc port-forward <pod> 5000 6000
```

Example output

```
Forwarding from 127.0.0.1:5000 -> 5000
Forwarding from [::1]:5000 -> 5000
Forwarding from 127.0.0.1:6000 -> 6000
Forwarding from [::1]:6000 -> 6000
```

- Use the following command to listen on port **8888** locally and forward to **5000** in the pod:

```
$ oc port-forward <pod> 8888:5000
```

Example output

```
Forwarding from 127.0.0.1:8888 -> 5000
Forwarding from [::1]:8888 -> 5000
```

- Use the following command to listen on a free port locally and forward to **5000** in the pod:

```
$ oc port-forward <pod> :5000
```

Example output

```
Forwarding from 127.0.0.1:42390 -> 5000
Forwarding from [::1]:42390 -> 5000
```

Or:

```
$ oc port-forward <pod> 0:5000
```

7.8.3. Protocol for initiating port forwarding from a client

Clients initiate port forwarding to a pod by issuing a request to the Kubernetes API server:

```
/proxy/nodes/<node_name>/portForward/<namespace>/<pod>
```

In the above URL:

- **<node_name>** is the FQDN of the node.
- **<namespace>** is the namespace of the target pod.
- **<pod>** is the name of the target pod.

For example:

```
/proxy/nodes/node123.openshift.com/portForward/myns/mypod
```

After sending a port forward request to the API server, the client upgrades the connection to one that supports multiplexed streams; the current implementation uses [Hypertext Transfer Protocol Version 2 \(HTTP/2\)](#).

The client creates a stream with the **port** header containing the target port in the pod. All data written to the stream is delivered via the kubelet to the target pod and port. Similarly, all data sent from the pod for that forwarded connection is delivered back to the same stream in the client.

The client closes all streams, the upgraded connection, and the underlying connection when it is finished with the port forwarding request.

7.9. USING SYSCTLS IN CONTAINERS

Sysctl settings are exposed through Kubernetes, allowing users to modify certain kernel parameters at runtime. Only sysctls that are namespaced can be set independently on pods. If a sysctl is not namespaced, called *node-level*, you must use another method of setting the sysctl, such as by using the Node Tuning Operator.

Network sysctls are a special category of sysctl. Network sysctls include:

- System-wide sysctls, for example **net.ipv4.ip_local_port_range**, that are valid for all networking. You can set these independently for each pod on a node.
- Interface-specific sysctls, for example **net.ipv4.conf.IFNAME.accept_local**, that only apply to a specific additional network interface for a given pod. You can set these independently for each additional network configuration. You set these by using a configuration in the **tuning-cni** after the network interfaces are created.

Moreover, only those sysctls considered *safe* are whitelisted by default; you can manually enable other *unsafe* sysctls on the node to be available to the user.

If you are setting the `sysctl` and it is node-level, you can find information on this procedure in the section [Using the Node Tuning Operator](#).

7.9.1. About sysctls

In Linux, the `sysctl` interface allows an administrator to modify kernel parameters at runtime. Parameters are available from the `/proc/sys/` virtual process file system. The parameters cover various subsystems, such as:

- kernel (common prefix: ***kernel.***)
- networking (common prefix: ***net.***)
- virtual memory (common prefix: ***vm.***)
- MDADM (common prefix: ***dev.***)

More subsystems are described in [Kernel documentation](#). To get a list of all parameters, run:

```
$ sudo sysctl -a
```

7.9.2. Namespaced and node-level sysctls

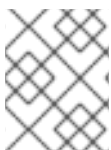
A number of `sysctls` are *namespaced* in the Linux kernels. This means that you can set them independently for each pod on a node. Being namespaced is a requirement for `sysctls` to be accessible in a pod context within Kubernetes.

The following `sysctls` are known to be namespaced:

- ***kernel.shm****
- ***kernel.msg****
- ***kernel.sem***
- ***fs.mqueue.****

Additionally, most of the `sysctls` in the ***net.**** group are known to be namespaced. Their namespace adoption differs based on the kernel version and distributor.

`Sysctls` that are not namespaced are called *node-level* and must be set manually by the cluster administrator, either by means of the underlying Linux distribution of the nodes, such as by modifying the `/etc/sysctls.conf` file, or by using a daemon set with privileged containers. You can use the Node Tuning Operator to set *node-level* `sysctls`.



NOTE

Consider marking nodes with special `sysctls` as tainted. Only schedule pods onto them that need those `sysctl` settings. Use the taints and toleration feature to mark the nodes.

7.9.3. Safe and unsafe sysctls

`Sysctls` are grouped into *safe* and *unsafe* `sysctls`.

For system-wide sysctls to be considered safe, they must be namespaced. A namespaced sysctl ensures there is isolation between namespaces and therefore pods. If you set a sysctl for one pod it must not add any of the following:

- Influence any other pod on the node
- Harm the node health
- Gain CPU or memory resources outside of the resource limits of a pod



NOTE

Being namespaced alone is not sufficient for the sysctl to be considered safe.

Any sysctl that is not added to the allowed list on OpenShift Container Platform is considered unsafe for OpenShift Container Platform.

Unsafe sysctls are not allowed by default. For system-wide sysctls the cluster administrator must manually enable them on a per-node basis. Pods with disabled unsafe sysctls are scheduled but do not launch.



NOTE

You cannot manually enable interface-specific unsafe sysctls.

OpenShift Container Platform adds the following system-wide and interface-specific safe sysctls to an allowed safe list:

Table 7.4. System-wide safe sysctls

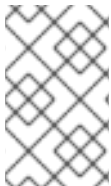
sysctl	Description
kernel.shm_rmid_forced	When set to 1 , all shared memory objects in current IPC namespace are automatically forced to use IPC_RMID. For more information, see shm_rmid_forced .
net.ipv4.ip_local_port_range	Defines the local port range that is used by TCP and UDP to choose the local port. The first number is the first port number, and the second number is the last local port number. If possible, it is better if these numbers have different parity (one even and one odd value). They must be greater than or equal to ip_unprivileged_port_start . The default values are 32768 and 60999 respectively. For more information, see ip_local_port_range .
net.ipv4.tcp_syncookies	When net.ipv4.tcp_syncookies is set, the kernel handles TCP SYN packets normally until the half-open connection queue is full, at which time, the SYN cookie functionality kicks in. This functionality allows the system to keep accepting valid connections, even if under a denial-of-service attack. For more information, see tcp_syncookies .
net.ipv4.ping_group_range	This restricts ICMP_PROTO datagram sockets to users in the group range. The default is 1 0 , meaning that nobody, not even root, can create ping sockets. For more information, see ping_group_range .

sysctl	Description
net.ipv4.ip_unprivileged_port_start	This defines the first unprivileged port in the network namespace. To disable all privileged ports, set this to 0 . Privileged ports must not overlap with the ip_local_port_range . For more information, see ip_unprivileged_port_start .
net.ipv4.ip_local_reserved_ports	Specify a range of comma-separated local ports that you want to reserve for applications or services.
net.ipv4.tcp_keepalive_time	Specify the interval in seconds before the first keepalive probe should be sent after a connection has become idle.
net.ipv4.tcp_fin_timeout	Specify the time in seconds that a connection remains in the FIN-WAIT-2 state before it is aborted.
net.ipv4.tcp_keepalive_intvl	Specify the interval in seconds between the keepalive probes. This value is multiplied by the tcp_keepalive_probes value to determine the total time required before it is decided that the connection is broken.
net.ipv4.tcp_keepalive_probes	Specify how many keepalive probes to send until it is determined that the connection is broken.

Table 7.5. Interface-specific safe sysctls

sysctl	Description
net.ipv4.conf.IFNAME.accept_redirects	Accept IPv4 ICMP redirect messages.
net.ipv4.conf.IFNAME.accept_source_route	Accept IPv4 packets with strict source route (SRR) option.
net.ipv4.conf.IFNAME.arp_accept	Define behavior for gratuitous ARP frames with an IPv4 address that is not already present in the ARP table: • 0 - Do not create new entries in the ARP table. • 1 - Create new entries in the ARP table.
net.ipv4.conf.IFNAME.arp_notify	Define mode for notification of IPv4 address and device changes.
net.ipv4.conf.IFNAME.disable_policy	Disable IPSEC policy (SPD) for this IPv4 interface.
net.ipv4.conf.IFNAME.secure_redirects	Accept ICMP redirect messages only to gateways listed in the interface's current gateway list.

sysctl	Description
net.ipv4.conf.IFNAME.send_redirects	Send redirects is enabled only if the node acts as a router. That is, a host should not send an ICMP redirect message. It is used by routers to notify the host about a better routing path that is available for a particular destination.
net.ipv6.conf.IFNAME.accept_ra	Accept IPv6 Router advertisements; autoconfigure using them. It also determines whether or not to transmit router solicitations. Router solicitations are transmitted only if the functional setting is to accept router advertisements.
net.ipv6.conf.IFNAME.accept_redirects	Accept IPv6 ICMP redirect messages.
net.ipv6.conf.IFNAME.accept_source_route	Accept IPv6 packets with SRR option.
net.ipv6.conf.IFNAME.arp_accept	Define behavior for gratuitous ARP frames with an IPv6 address that is not already present in the ARP table: • 0 - Do not create new entries in the ARP table. • 1 - Create new entries in the ARP table.
net.ipv6.conf.IFNAME.arp_notify	Define mode for notification of IPv6 address and device changes.
net.ipv6.neigh.IFNAME.base_reachable_time_ms	This parameter controls the hardware address to IP mapping lifetime in the neighbour table for IPv6.
net.ipv6.neigh.IFNAME.retrans_time_ms	Set the retransmit timer for neighbor discovery messages.

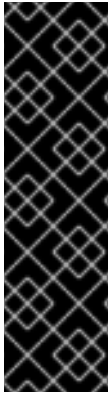


NOTE

When setting these values using the **tuning** CNI plugin, use the value **IFNAME** literally. The interface name is represented by the **IFNAME** token, and is replaced with the actual name of the interface at runtime.

7.9.4. Updating the interface-specific safe sysctls list

OpenShift Container Platform includes a predefined list of safe interface-specific **sysctls**. You can modify this list by updating the **cni-sysctl-allowlist** in the **openshift-multus** namespace.



IMPORTANT

The support for updating the interface-specific safe sysctls list is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

Follow this procedure to modify the predefined list of safe **sysctls**. This procedure describes how to extend the default allow list.

Procedure

1. View the existing predefined list by running the following command:

```
$ oc get cm -n openshift-multus cni-sysctl-allowlist -oyaml
```

Expected output

```
apiVersion: v1
data:
  allowlist.conf: |-
    ^net.ipv4.conf.IFNAME.accept_redirects$
    ^net.ipv4.conf.IFNAME.accept_source_route$
    ^net.ipv4.conf.IFNAME.arp_accept$
    ^net.ipv4.conf.IFNAME.arp_notify$
    ^net.ipv4.conf.IFNAME.disable_policy$
    ^net.ipv4.conf.IFNAME.secure_redirects$
    ^net.ipv4.conf.IFNAME.send_redirects$
    ^net.ipv6.conf.IFNAME.accept_ra$
    ^net.ipv6.conf.IFNAME.accept_redirects$
    ^net.ipv6.conf.IFNAME.accept_source_route$
    ^net.ipv6.conf.IFNAME.arp_accept$
    ^net.ipv6.conf.IFNAME.arp_notify$
    ^net.ipv6.neigh.IFNAME.base_reachable_time_ms$
    ^net.ipv6.neigh.IFNAME.retrans_time_ms$
kind: ConfigMap
metadata:
  annotations:
    kubernetes.io/description: |
      Sysctl allowlist for nodes.
    release.openshift.io/version: 4.20.0-0.nightly-2022-11-16-003434
  creationTimestamp: "2022-11-17T14:09:27Z"
  name: cni-sysctl-allowlist
  namespace: openshift-multus
  resourceVersion: "2422"
  uid: 96d138a3-160e-4943-90ff-6108fa7c50c3
```

2. Edit the list by using the following command:

```
$ oc edit cm -n openshift-multus cni-sysctl-allowlist -oyaml
```



For example, to allow you to be able to implement stricter reverse path forwarding you need to add `^net.ipv4.conf.IFNAME.rp_filter$` and `^net.ipv6.conf.IFNAME.rp_filter$` to the list as shown here:

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  allowlist.conf: |-
    ^net.ipv4.conf.IFNAME.accept_redirects$
    ^net.ipv4.conf.IFNAME.accept_source_route$
    ^net.ipv4.conf.IFNAME.arp_accept$
    ^net.ipv4.conf.IFNAME.arp_notify$
    ^net.ipv4.conf.IFNAME.disable_policy$
    ^net.ipv4.conf.IFNAME.secure_redirects$
    ^net.ipv4.conf.IFNAME.send_redirects$
    ^net.ipv4.conf.IFNAME.rp_filter$
    ^net.ipv6.conf.IFNAME.accept_ra$
    ^net.ipv6.conf.IFNAME.accept_redirects$
    ^net.ipv6.conf.IFNAME.accept_source_route$
    ^net.ipv6.conf.IFNAME.arp_accept$
    ^net.ipv6.conf.IFNAME.arp_notify$
    ^net.ipv6.neigh.IFNAME.base_reachable_time_ms$
    ^net.ipv6.neigh.IFNAME.retrans_time_ms$
    ^net.ipv6.conf.IFNAME.rp_filter$
```

3. Save the changes to the file and exit.



NOTE

The removal of **sysctls** is also supported. Edit the file, remove the **sysctl** or **sysctls** then save the changes and exit.

Verification

Follow this procedure to enforce stricter reverse path forwarding for IPv4. For more information on reverse path forwarding see [Reverse Path Forwarding](#) .

1. Create a network attachment definition, such as **reverse-path-fwd-example.yaml**, with the following content:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: tuningnad
  namespace: default
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "tuningnad",
    "plugins": [{
      "type": "bridge"
```

```

    },
    {
      "type": "tuning",
      "sysctl": {
        "net.ipv4.conf.IFNAME.rp_filter": "1"
      }
    }
  ]
}'

```

2. Apply the yaml by running the following command:

```
$ oc apply -f reverse-path-fwd-example.yaml
```

Example output

```
networkattachmentdefinition.k8s.cni.cncf.io/tuningnad created
```

3. Create a pod such as **examplepod.yaml** using the following YAML:

```

apiVersion: v1
kind: Pod
metadata:
  name: example
  labels:
    app: httpd
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: tuningnad ❶
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: httpd
      image: 'image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest'
      ports:
        - containerPort: 8080
      securityContext:
        allowPrivilegeEscalation: false
      capabilities:
        drop:
          - ALL

```

- ❶ Specify the name of the configured **NetworkAttachmentDefinition**.

4. Apply the yaml by running the following command:

```
$ oc apply -f examplepod.yaml
```

5. Verify that the pod is created by running the following command:

```
$ oc get pod
```

Example output

```
NAME      READY  STATUS   RESTARTS  AGE
example   1/1    Running  0          47s
```

- Log in to the pod by running the following command:

```
$ oc rsh example
```

- Verify the value of the configured sysctl flag. For example, find the value **net.ipv4.conf.net1.rp_filter** by running the following command:

```
sh-4.4# sysctl net.ipv4.conf.net1.rp_filter
```

Expected output

```
net.ipv4.conf.net1.rp_filter = 1
```

Additional resources

- [Linux networking documentation](#)

7.9.5. Starting a pod with safe sysctls

You can set sysctls on pods using the pod's **securityContext**. The **securityContext** applies to all containers in the same pod.

Safe sysctls are allowed by default.

This example uses the pod **securityContext** to set the following safe sysctls:

- kernel.shm_rmid_forced**
- net.ipv4.ip_local_port_range**
- net.ipv4.tcp_syncookies**
- net.ipv4.ping_group_range**



WARNING

To avoid destabilizing your operating system, modify sysctl parameters only after you understand their effects.

Use this procedure to start a pod with the configured sysctl settings.



NOTE

In most cases you modify an existing pod definition and add the **securityContext** spec.

Procedure

1. Create a YAML file **sysctl_pod.yaml** that defines an example pod and add the **securityContext** spec, as shown in the following example:

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example
  namespace: default
spec:
  containers:
  - name: podexample
    image: centos
    command: ["bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000 1
      runAsGroup: 3000 2
      allowPrivilegeEscalation: false 3
      capabilities: 4
      drop: ["ALL"]
    securityContext:
      runAsNonRoot: true 5
      seccompProfile: 6
      type: RuntimeDefault
    sysctls:
      - name: kernel.shm_rmid_forced
        value: "1"
      - name: net.ipv4.ip_local_port_range
        value: "32770 60666"
      - name: net.ipv4.tcp_syncookies
        value: "0"
      - name: net.ipv4.ping_group_range
        value: "0 200000000"
```

- 1 **runAsUser** controls which user ID the container is run with.
- 2 **runAsGroup** controls which primary group ID the containers is run with.
- 3 **allowPrivilegeEscalation** determines if a pod can request to allow privilege escalation. If unspecified, it defaults to true. This boolean directly controls whether the **no_new_privs** flag gets set on the container process.
- 4 **capabilities** permit privileged actions without giving full root access. This policy ensures all capabilities are dropped from the pod.
- 5 **runAsNonRoot: true** requires that the container will run with a user with any UID other than 0.
- 6 **RuntimeDefault** enables the default seccomp profile for a pod or container workload.

2. Create the pod by running the following command:

```
$ oc apply -f sysctl_pod.yaml
```

3. Verify that the pod is created by running the following command:

```
$ oc get pod
```

Example output

```
NAME          READY  STATUS   RESTARTS  AGE
sysctl-example 1/1    Running   0          14s
```

4. Log in to the pod by running the following command:

```
$ oc rsh sysctl-example
```

5. Verify the values of the configured sysctl flags. For example, find the value **kernel.shm_rmid_forced** by running the following command:

```
sh-4.4# sysctl kernel.shm_rmid_forced
```

Expected output

```
kernel.shm_rmid_forced = 1
```

7.9.6. Starting a pod with unsafe sysctls

A pod with unsafe sysctls fails to launch on any node unless the cluster administrator explicitly enables unsafe sysctls for that node. As with node-level sysctls, use the taints and toleration feature or labels on nodes to schedule those pods onto the right nodes.

The following example uses the pod **securityContext** to set a safe sysctl **kernel.shm_rmid_forced** and two unsafe sysctls, **net.core.somaxconn** and **kernel.msgmax**. There is no distinction between *safe* and *unsafe* sysctls in the specification.



WARNING

To avoid destabilizing your operating system, modify sysctl parameters only after you understand their effects.

The following example illustrates what happens when you add safe and unsafe sysctls to a pod specification:

Procedure

1. Create a YAML file **sysctl-example-unsafe.yaml** that defines an example pod and add the **securityContext** specification, as shown in the following example:

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example-unsafe
spec:
  containers:
    - name: podexample
      image: centos
      command: ["bin/bash", "-c", "sleep INF"]
      securityContext:
        runAsUser: 2000
        runAsGroup: 3000
        allowPrivilegeEscalation: false
        capabilities:
          drop: ["ALL"]
      securityContext:
        runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault
      sysctls:
        - name: kernel.shm_rmid_forced
          value: "0"
        - name: net.core.somaxconn
          value: "1024"
        - name: kernel.msgmax
          value: "65536"
```

2. Create the pod using the following command:

```
$ oc apply -f sysctl-example-unsafe.yaml
```

3. Verify that the pod is scheduled but does not deploy because unsafe sysctls are not allowed for the node using the following command:

```
$ oc get pod
```

Example output

NAME	READY	STATUS	RESTARTS	AGE
sysctl-example-unsafe	0/1	SysctlForbidden	0	14s

7.9.7. Enabling unsafe sysctls

A cluster administrator can allow certain unsafe sysctls for very special situations such as high performance or real-time application tuning.

If you want to use unsafe sysctls, a cluster administrator must enable them individually for a specific type of node. The sysctls must be namespaced.

You can further control which sysctls are set in pods by specifying lists of sysctls or sysctl patterns in the **allowedUnsafeSysctls** field of the Security Context Constraints.

- The **allowedUnsafeSysctls** option controls specific needs such as high performance or real-time application tuning.



WARNING

Due to their nature of being unsafe, the use of unsafe sysctls is at-your-own-risk and can lead to severe problems, such as improper behavior of containers, resource shortage, or breaking a node.

Procedure

1. List existing MachineConfig objects for your OpenShift Container Platform cluster to decide how to label your machine config by running the following command:

```
$ oc get machineconfigpool
```

Example output

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT		
DEGRADEDMACHINECOUNT	AGE			
master	rendered-master-bfb92f0cd1684e54d8e234ab7423cc96	True	False	False
3	3	3	0	42m
worker	rendered-worker-21b6cb9a0f8919c88caf39db80ac1fce	True	False	False
3	3	3	0	42m

2. Add a label to the machine config pool where the containers with the unsafe sysctls will run by running the following command:

```
$ oc label machineconfigpool worker custom-kubelet=sysctl
```

3. Create a YAML file **set-sysctl-worker.yaml** that defines a **KubeletConfig** custom resource (CR):

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-kubelet
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: sysctl 1
  kubeletConfig:
    allowedUnsafeSysctls: 2
    - "kernel.msg*"
    - "net.core.somaxconn"
```

- 1** Specify the label from the machine config pool.

- 2 List the unsafe sysctls you want to allow.

4. Create the object by running the following command:

```
$ oc apply -f set-sysctl-worker.yaml
```

5. Wait for the Machine Config Operator to generate the new rendered configuration and apply it to the machines by running the following command:

```
$ oc get machineconfigpool worker -w
```

After some minutes the **UPDATING** status changes from True to False:

	NAME	CONFIG		UPDATED	UPDATING	DEGRADED
	MACHINECOUNT	READYMACHINECOUNT		UPDATEDMACHINECOUNT		
	DEGRADEDMACHINECOUNT	AGE				
worker	rendered-worker-f1704a00fc6f30d3a7de9a15fd68a800	False	True	False		
3	2	2	0	71m		
worker	rendered-worker-f1704a00fc6f30d3a7de9a15fd68a800	False	True	False		
3	2	3	0	72m		
worker	rendered-worker-0188658afe1f3a183ec8c4f14186f4d5	True	False	False		
3	3	3	0	72m		

6. Create a YAML file **sysctl-example-safe-unsafe.yaml** that defines an example pod and add the **securityContext** spec, as shown in the following example:

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example-safe-unsafe
spec:
  containers:
  - name: podexample
    image: centos
    command: ["bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000
      runAsGroup: 3000
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
    securityContext:
      runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault
      sysctls:
      - name: kernel.shm_rmid_forced
        value: "0"
      - name: net.core.somaxconn
        value: "1024"
      - name: kernel.msgmax
        value: "65536"
```

7. Create the pod by running the following command:

```
$ oc apply -f sysctl-example-safe-unsafe.yaml
```

Expected output

```
Warning: would violate PodSecurity "restricted:latest": forbidden sysctls
(net.core.somaxconn, kernel.msgmax)
pod/sysctl-example-safe-unsafe created
```

8. Verify that the pod is created by running the following command:

```
$ oc get pod
```

Example output

```
NAME                READY STATUS  RESTARTS  AGE
sysctl-example-safe-unsafe  1/1   Running  0         19s
```

9. Log in to the pod by running the following command:

```
$ oc rsh sysctl-example-safe-unsafe
```

10. Verify the values of the configured sysctl flags. For example, find the value **net.core.somaxconn** by running the following command:

```
sh-4.4# sysctl net.core.somaxconn
```

Expected output

```
net.core.somaxconn = 1024
```

The unsafe sysctl is now allowed and the value is set as defined in the **securityContext** spec of the updated pod specification.

7.9.8. Additional resources

- [Configuring system controls by using the tuning CNI](#)
- [Using the Node Tuning Operator](#)

7.10. ACCESSING FASTER BUILDS WITH /DEV/FUSE

You can configure your pods with the **/dev/fuse** device to enable faster and more efficient container image builds, particularly for unprivileged users. This device allows unprivileged pods to mount overlay filesystems, which can be leveraged by tools like Podman.

7.10.1. Configuring /dev/fuse for unprivileged builds in pods

By exposing the **/dev/fuse** device to an unprivileged pod, you grant it the capability to perform Filesystem in Userspace (FUSE) mounts. This is achieved by adding the **io.kubernetes.cri-o.Devices:**

"/dev/fuse" annotation to your pod definition. This setup allows an unprivileged user within the pod to use tools like **podman** with storage drivers such as **fuse-overlayfs** by mimicking privileged build capabilities in a secure and efficient manner without granting full privileged access to the pod.

Procedure

1. Define the pod with **/dev/fuse** access:
 - a. Create a YAML file named **fuse-builder-pod.yaml** with the following content:

```
apiVersion: v1
kind: Pod
metadata:
  name: fuse-builder-pod
  annotations:
    io.kubernetes.cri-o.Devices: "/dev/fuse"
spec:
  containers:
  - name: build-container
    image: quay.io/podman/stable
    command: ["/bin/sh", "-c"]
    args: ["echo 'Container is running. Use oc exec to get a shell.'; sleep infinity"]
    securityContext:
      runAsUser: 1000
```

where:

io.kubernetes.cri-o.Devices

The **io.kubernetes.cri-o.Devices: "/dev/fuse"** annotation makes the FUSE device available.

image

This annotation specifies a container that uses an image that includes **podman** (for example, **quay.io/podman/stable**).

args

This command keeps the container running so you can **exec** into it.

securityContext

This annotation specifies a **securityContext** that runs the container as an unprivileged user (for example, **runAsUser: 1000**).



NOTE

Depending on your cluster's Security Context Constraints (SCCs) or other policies, you might need to further adjust the **securityContext** specification, for example, by allowing specific capabilities if **/dev/fuse** alone is not sufficient for **fuse-overlayfs** to operate.

- b. Create the pod by running the following command:

```
$ oc apply -f fuse-builder-pod.yaml
```

2. Verify that the pod is running:

```
$ oc get pods fuse-builder-pod
```

3. Access the pod and prepare the build environment:

After the **fuse-builder-pod** pod is in the **Running** state, open a shell session into the **build-container** environment:

```
$ oc exec -ti fuse-builder-pod -- /bin/bash
```

You are now inside the container. Because the default working directory might not be writable by the unprivileged user, change to a writable directory like **/tmp**:

```
$ cd /tmp
```

```
$ pwd
```

```
/tmp
```

4. Create a dockerfile and build an image using Podman:

Inside the pod's shell and within the **/tmp** directory, you can now create a **Dockerfile** and use **podman** to build a container image. If **fuse-overlayfs** is the default or configured storage driver, Podman is able to leverage **fuse-overlayfs** because of the available **/dev/fuse** device.

- a. Create a sample **Dockerfile**:

```
$ cat > Dockerfile <<EOF
FROM registry.access.redhat.com/ubi9/ubi-minimal
RUN microdnf install -y findutils && microdnf clean all
RUN echo "This image was built inside a pod with /dev/fuse by user $(id -u)" >
/app/build_info.txt
COPY Dockerfile /app/Dockerfile_copied
WORKDIR /app
CMD ["sh", "-c", "cat /app/build_info.txt && echo '--- Copied Dockerfile ---' && cat
/app/Dockerfile_copied"]
EOF
```

- b. Build the image using **podman**. The **-t** flag tags the image:

```
$ podman build -t my-fuse-built-image:latest .
```

You should see Podman executing the build steps.

5. Optional: Test the built image:

Still inside the **fuse-builder-pod**, you can run a container from the image you just built to test it:

```
$ podman run --rm my-fuse-built-image:latest
```

This should output the content of the **/app/build_info.txt** file and the copied Dockerfile.

6. Exit the pod and clean up:

- a. After you are done, exit the shell session in the pod:

```
$ exit
```

- b. Delete the pod if it's no longer needed:

```
$ oc delete pod fuse-builder-pod
```

- c. Remove the local YAML file:

```
$ rm fuse-builder-pod.yaml
```

CHAPTER 8. WORKING WITH CLUSTERS

8.1. VIEWING SYSTEM EVENT INFORMATION IN OPENSIFT CONTAINER PLATFORM CLUSTERS

You can view events in OpenShift Container Platform, which are based on events that happen to API objects in an OpenShift Container Platform cluster.

8.1.1. Understanding events

Review the following information to learn how OpenShift Container Platform uses *events* to record information about real-world events in a resource-agnostic manner. Events also allow developers and administrators to consume information about system components in a unified way.

8.1.2. Viewing events using the CLI

You can get a list of events in a given project by using the CLI.

Procedure

- View events in a project by using a command similar to the following:

```
$ oc get events [-n <project>]
```

where:

project

Specifies the name of the project.

For example:

```
$ oc get events -n openshift-config
```

Example output

```
LAST SEEN   TYPE      REASON              OBJECT                               MESSAGE
97m         Normal    Scheduled            pod/dapi-env-test-pod               Successfully assigned
openshift-config/dapi-env-test-pod to ip-10-0-171-202.ec2.internal
97m         Normal    Pulling             pod/dapi-env-test-pod               pulling image
"gcr.io/google_containers/busybox"
97m         Normal    Pulled              pod/dapi-env-test-pod               Successfully pulled image
"gcr.io/google_containers/busybox"
97m         Normal    Created             pod/dapi-env-test-pod               Created container
9m5s        Warning   FailedCreatePodSandBox pod/dapi-volume-test-pod           Failed create
pod sandbox: rpc error: code = Unknown desc = failed to create pod network sandbox
k8s_dapi-volume-test-pod_openshift-config_6bc60c1f-452e-11e9-9140-
0eec59c23068_0(748c7a40db3d08c07fb4f9eba774bd5effe5f0d5090a242432a73eee66ba9e22
): Multus: Err adding pod to network "ovn-kubernetes": cannot set "ovn-kubernetes" ifname to
"eth0": no netns: failed to Statfs "/proc/33366/ns/net": no such file or directory
8m31s      Normal    Scheduled            pod/dapi-volume-test-pod           Successfully assigned
openshift-config/dapi-volume-test-pod to ip-10-0-171-202.ec2.internal
#...
```

- View events in your project from the OpenShift Container Platform console:
 1. Launch the OpenShift Container Platform console.
 2. Click **Home** → **Events** and select your project.
 3. Move to resource that you want to see events. For example: **Home** → **Projects** → <project-name> → <resource-name>.

Many objects, such as pods and deployments, also have an **Events** tab, which shows events related to that object.

8.1.3. List of events

Review the information in this section to learn about OpenShift Container Platform events.

Table 8.1. Configuration events

Name	Description
FailedValidation	Failed pod configuration validation.

Table 8.2. Container events

Name	Description
BackOff	Back-off restarting failed the container.
Created	Container created.
Failed	Pull/Create/Start failed.
Killing	Killing the container.
Started	Container started.
Preempting	Preempting other pods.
ExceededGrace Period	Container runtime did not stop the pod within specified grace period.

Table 8.3. Health events

Name	Description
Unhealthy	Container is unhealthy.

Table 8.4. Image events

Name	Description
BackOff	Back off Ctr Start, image pull.
ErrImageNeverPull	The image's NeverPull Policy is violated.
Failed	Failed to pull the image.
InspectFailed	Failed to inspect the image.
Pulled	Successfully pulled the image or the container image is already present on the machine.
Pulling	Pulling the image.

Table 8.5. Image Manager events

Name	Description
FreeDiskSpaceFailed	Free disk space failed.
InvalidDiskCapacity	Invalid disk capacity.

Table 8.6. Node events

Name	Description
FailedMount	Volume mount failed.
HostNetworkNotSupported	Host network not supported.
HostPortConflict	Host/port conflict.
KubeletSetupFailed	Kubelet setup failed.
NilShaper	Undefined shaper.
NodeNotReady	Node is not ready.
NodeNotSchedulable	Node is not schedulable.

Name	Description
NodeReady	Node is ready.
NodeSchedulable	Node is schedulable.
NodeSelectorMismatching	Node selector mismatch.
OutOfDisk	Out of disk.
Rebooted	Node rebooted.
Starting	Starting kubelet.
FailedAttachVolume	Failed to attach volume.
FailedDetachVolume	Failed to detach volume.
VolumeResizeFailed	Failed to expand/reduce volume.
VolumeResizeSuccessful	Successfully expanded/reduced volume.
FileSystemResizeFailed	Failed to expand/reduce file system.
FileSystemResizeSuccessful	Successfully expanded/reduced file system.
FailedUnMount	Failed to unmount volume.
FailedMapVolume	Failed to map a volume.
FailedUnmapDevice	Failed unmaped device.
AlreadyMountedVolume	Volume is already mounted.
SuccessfulDetachVolume	Volume is successfully detached.

Name	Description
SuccessfulMountVolume	Volume is successfully mounted.
SuccessfulUnmountVolume	Volume is successfully unmounted.
ContainerGCFailed	Container garbage collection failed.
ImageGCFailed	Image garbage collection failed.
FailedNodeAllocatableEnforcement	Failed to enforce System Reserved Cgroup limit.
NodeAllocatableEnforced	Enforced System Reserved Cgroup limit.
UnsupportedMountOption	Unsupported mount option.
SandboxChanged	Pod sandbox changed.
FailedCreatePodSandbox	Failed to create pod sandbox.
FailedPodSandboxStatus	Failed pod sandbox status.

Table 8.7. Pod worker events

Name	Description
FailedSync	Pod sync failed.

Table 8.8. System Events

Name	Description
SystemOOM	There is an OOM (out of memory) situation on the cluster.

Table 8.9. Pod events

Name	Description
FailedKillPod	Failed to stop a pod.
FailedCreatePodContainer	Failed to create a pod container.
Failed	Failed to make pod data directories.
NetworkNotReady	Network is not ready.
FailedCreate	Error creating: <error-msg> .
SuccessfulCreate	Created pod: <pod-name> .
FailedDelete	Error deleting: <error-msg> .
SuccessfulDelete	Deleted pod: <pod-id> .

Table 8.10. Horizontal Pod AutoScaler events

Name	Description
SelectorRequired	Selector is required.
InvalidSelector	Could not convert selector into a corresponding internal selector object.
FailedGetObjectMetric	HPA was unable to compute the replica count.
InvalidMetricSourceType	Unknown metric source type.
ValidMetricFound	HPA was able to successfully calculate a replica count.
FailedConvertHPA	Failed to convert the given HPA.
FailedGetScale	HPA controller was unable to get the target's current scale.

Name	Description
SucceededGetScale	HPA controller was able to get the target's current scale.
FailedComputeMetricsReplicas	Failed to compute desired number of replicas based on listed metrics.
FailedRescale	New size: <size> ; reason: <msg> ; error: <error-msg> .
SuccessfulRescale	New size: <size> ; reason: <msg> .
FailedUpdateStatus	Failed to update status.

Table 8.11. Volume events

Name	Description
FailedBinding	There are no persistent volumes available and no storage class is set.
VolumeMismatch	Volume size or class is different from what is requested in claim.
VolumeFailedRecycle	Error creating recycler pod.
VolumeRecycled	Occurs when volume is recycled.
RecyclerPod	Occurs when pod is recycled.
VolumeDelete	Occurs when volume is deleted.
VolumeFailedDelete	Error when deleting the volume.
ExternalProvisioning	Occurs when volume for the claim is provisioned either manually or via external software.
ProvisioningFailed	Failed to provision volume.

Name	Description
ProvisioningCleanupFailed	Error cleaning provisioned volume.
ProvisioningSucceeded	Occurs when the volume is provisioned successfully.
WaitForFirstConsumer	Delay binding until pod scheduling.

Table 8.12. Lifecycle hooks

Name	Description
FailedPostStartHook	Handler failed for pod start.
FailedPreStopHook	Handler failed for pre-stop.
UnfinishedPreStopHook	Pre-stop hook unfinished.

Table 8.13. Deployments

Name	Description
DeploymentCancellationFailed	Failed to cancel deployment.
DeploymentCancelled	Canceled deployment.
DeploymentCreated	Created new replication controller.
IngressIPRangeFull	No available Ingress IP to allocate to service.

Table 8.14. Scheduler events

Name	Description
FailedScheduling	Failed to schedule pod: <pod-namespace>/<pod-name> . This event is raised for multiple reasons, for example: AssumePodVolumes failed, Binding rejected etc.
Preempted	By <preemptor-namespace>/<preemptor-name> on node <node-name> .
Scheduled	Successfully assigned <pod-name> to <node-name> .

Table 8.15. Daemon set events

Name	Description
SelectingAll	This daemon set is selecting all pods. A non-empty selector is required.
FailedPlacement	Failed to place pod on <node-name> .
FailedDaemonPod	Found failed daemon pod <pod-name> on node <node-name> , will try to kill it.

Table 8.16. LoadBalancer service events

Name	Description
CreatingLoadBalancerFailed	Error creating load balancer.
DeletingLoadBalancer	Deleting load balancer.
EnsuringLoadBalancer	Ensuring load balancer.
EnsuredLoadBalancer	Ensured load balancer.
UnavailableLoadBalancer	There are no available nodes for LoadBalancer service.
LoadBalancerSourceRanges	Lists the new LoadBalancerSourceRanges . For example, <old-source-range> → <new-source-range> .
LoadbalancerIP	Lists the new IP address. For example, <old-ip> → <new-ip> .

Name	Description
ExternalIP	Lists external IP address. For example, Added: <external-ip> .
UID	Lists the new UID. For example, <old-service-uid> → <new-service-uid> .
ExternalTrafficPolicy	Lists the new ExternalTrafficPolicy . For example, <old-policy> → <new-policy> .
HealthCheckNodePort	Lists the new HealthCheckNodePort . For example, <old-node-port> → new-node-port> .
UpdatedLoadBalancer	Updated load balancer with new hosts.
LoadBalancerUpdateFailed	Error updating load balancer with new hosts.
DeletingLoadBalancer	Deleting load balancer.
DeletingLoadBalancerFailed	Error deleting load balancer.
DeletedLoadBalancer	Deleted load balancer.

8.2. ESTIMATING THE NUMBER OF PODS YOUR OPENSIFT CONTAINER PLATFORM NODES CAN HOLD

As a cluster administrator, you can use the OpenShift Cluster Capacity Tool to view the number of pods that can be scheduled in your cluster. This allows you to increase the current resources before they become exhausted and to ensure any future pods can be scheduled. This capacity comes from an individual node host in a cluster, and includes CPU, memory, disk space, and others.

8.2.1. Understanding the OpenShift Cluster Capacity Tool

Review the following information to learn how to use the OpenShift Cluster Capacity Tool to simulate a sequence of scheduling decisions that determine how many instances of an input pod can be scheduled on the cluster before the cluster is exhausted of resources.



NOTE

The remaining allocatable capacity is a rough estimation, because it does not count all of the resources being distributed among nodes. It analyzes only the remaining resources and estimates the available capacity that is still consumable in terms of the number of instances of a pod with given requirements that can be scheduled in a cluster.

Also, pods might only have scheduling support on particular sets of nodes based on its selection and affinity criteria. As a result, the estimation of which remaining pods a cluster can schedule can be difficult.

You can run the OpenShift Cluster Capacity Tool as a stand-alone utility from the command line, or as a job in a pod inside an OpenShift Container Platform cluster. Running the tool as job inside of a pod enables you to run it multiple times without intervention.

8.2.2. Running the OpenShift Cluster Capacity Tool on the command line

You can run the OpenShift Cluster Capacity Tool from the command line to estimate the number of pods that can be scheduled onto your cluster.

You create a sample pod spec file, which the tool uses for estimating resource usage. The pod spec specifies its resource requirements as **limits** or **requests**. The cluster capacity tool takes the pod's resource requirements into account for its estimation analysis.

Prerequisites

1. Run the OpenShift Cluster Capacity Tool, which is available as a container image from the Red Hat Ecosystem Catalog. See the link in the "Additional resources" section.
2. Create a sample pod spec file:
 - a. Create a YAML file similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  securityContext:
    runAsNonRoot: true
  seccompProfile:
    type: RuntimeDefault
  containers:
  - name: php-redis
    image: gcr.io/google-samples/gb-frontend:v4
    imagePullPolicy: Always
    resources:
      limits:
        cpu: 150m
        memory: 100Mi
      requests:
        cpu: 150m
        memory: 100Mi
```

```
securityContext:
  allowPrivilegeEscalation: false
capabilities:
  drop: [ALL]
```

- b. Create the cluster role:

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

For example:

```
$ oc create -f pod-spec.yaml
```

Procedure

1. From the terminal, log in to the Red Hat Registry:

```
$ podman login registry.redhat.io
```

2. Pull the cluster capacity tool image:

```
$ podman pull registry.redhat.io/openshift4/ose-cluster-capacity
```

3. Run the cluster capacity tool:

```
$ podman run -v $HOME/.kube:/kube:Z -v $(pwd):/cc:Z ose-cluster-capacity \
/bin/cluster-capacity --kubeconfig /kube/config --<pod_spec>.yaml /cc/<pod_spec>.yaml \
--verbose
```

where:

<pod_spec>.yaml

Specifies the pod spec to use.

verbose

Outputs a detailed description of how many pods can be scheduled on each node in the cluster.

Example output

```
small-pod pod requirements:
- CPU: 150m
- Memory: 100Mi
```

The cluster can schedule 88 instance(s) of the pod small-pod.

```
Termination reason: Unschedulable: 0/5 nodes are available: 2 Insufficient cpu,
3 node(s) had taint {node-role.kubernetes.io/master: }, that the pod didn't
tolerate.
```

```
Pod distribution among nodes:
```

```
small-pod
- 192.168.124.214: 45 instance(s)
- 192.168.124.120: 43 instance(s)
```

In the above example, the number of estimated pods that can be scheduled onto the cluster is 88.

8.2.3. Running the OpenShift Cluster Capacity Tool as a job inside a pod

You can run the OpenShift Cluster Capacity Tool as a job inside of a pod by using a **ConfigMap** object. This allows you to run the tool multiple times without needing user intervention.

Prerequisites

- Download and install the OpenShift Cluster Capacity Tool from the **cluster-capacity** repository. See the link in the "Additional resources" section.

Procedure

- Create the cluster role:
 - Create a YAML file similar to the following:

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cluster-capacity-role
rules:
- apiGroups: [""]
  resources: ["pods", "nodes", "persistentvolumeclaims", "persistentvolumes", "services",
"replicationcontrollers"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["apps"]
  resources: ["replicasets", "statefulsets"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["policy"]
  resources: ["poddisruptionbudgets"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["storage.k8s.io"]
  resources: ["storageclasses"]
  verbs: ["get", "watch", "list"]
```

- Create the cluster role by running the following command:

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

For example:

```
$ oc create sa cluster-capacity-sa
```

- Create the service account:

```
$ oc create sa cluster-capacity-sa -n default
```

3. Add the role to the service account:

```
$ oc adm policy add-cluster-role-to-user cluster-capacity-role \
  system:serviceaccount:<namespace>:cluster-capacity-sa
```

where:

<namespace>

Specifies the namespace where the pod is located.

4. Define and create the pod spec:

- a. Create a YAML file similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: php-redis
    image: gcr.io/google-samples/gb-frontend:v4
    imagePullPolicy: Always
    resources:
      limits:
        cpu: 150m
        memory: 100Mi
      requests:
        cpu: 150m
        memory: 100Mi
    securityContext:
      allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
```

- b. Create the pod by running the following command:

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

For example:

```
$ oc create -f pod.yaml
```

5. Created a config map object by running the following command:

```
$ oc create configmap cluster-capacity-configmap \
  --from-file=pod.yaml=pod.yaml
```

The cluster capacity analysis is mounted in a volume using a config map object named **cluster-capacity-configmap** to mount the input pod spec file **pod.yaml** into a volume **test-volume** at the path **/test-pod**.

6. Create the job using the below example of a job specification file:
 - a. Create a YAML file similar to the following:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: cluster-capacity-job
spec:
  parallelism: 1
  completions: 1
  template:
    metadata:
      name: cluster-capacity-pod
    spec:
      containers:
        - name: cluster-capacity
          image: openshift/origin-cluster-capacity
          imagePullPolicy: "Always"
          volumeMounts:
            - mountPath: /test-pod
              name: test-volume
          env:
            - name: CC_INCLUSTER
              value: "true"
          command:
            - "/bin/sh"
            - "-ec"
            - |
              /bin/cluster-capacity --podspec=/test-pod/pod.yaml --verbose
          restartPolicy: "Never"
          serviceAccountName: cluster-capacity-sa
      volumes:
        - name: test-volume
          configMap:
            name: cluster-capacity-configmap
```

where:

spec.template.spec.containers.env

Specifies a required environment variable that indicates the Cluster Capacity Tool is running inside a cluster as a pod.

The **pod.yaml** key of the **ConfigMap** object is the same as the **Pod** spec file name, though it is not required. By doing this, the input pod spec file can be accessed inside the pod as **/test-pod/pod.yaml**.

- b. Run the cluster capacity image as a job in a pod by running the following command:

```
$ oc create -f cluster-capacity-job.yaml
```

Verification

1. Check the job logs to find the number of pods that can be scheduled in the cluster:

```
$ oc logs jobs/cluster-capacity-job
```

Example output

```
small-pod pod requirements:
```

- CPU: 150m
- Memory: 100Mi

```
The cluster can schedule 52 instance(s) of the pod small-pod.
```

```
Termination reason: Unschedulable: No nodes are available that match all of the
following predicates:: Insufficient cpu (2).
```

```
Pod distribution among nodes:
```

```
small-pod
```

- 192.168.124.214: 26 instance(s)
- 192.168.124.120: 26 instance(s)

8.2.4. Additional resources

- [OpenShift Cluster Capacity Tool](#)
- [cluster-capacity repository](#)

8.3. RESTRICT RESOURCE CONSUMPTION WITH LIMIT RANGES

You can use limit ranges to restrict resource consumption for specific objects in a project.

By default, containers run with unbounded compute resources on an OpenShift Container Platform cluster.

You can configure resource consumption for the following objects:

- pods and containers: You can set minimum and maximum requirements for CPU and memory for pods and their containers.
- Image streams: You can set limits on the number of images and tags in an **ImageStream** object.
- Images: You can limit the size of images that can be pushed to an internal registry.
- Persistent volume claims (PVC): You can restrict the size of the PVCs that can be requested.

If a pod does not meet the constraints imposed by the limit range, the pod cannot be created in the namespace.

8.3.1. About limit ranges

You can set specific resource limits for a pod, container, image, image stream, or persistent volume claim (PVC) in a specific project by defining a **LimitRange** object. A limit range allows you to restrict resource consumption in that project.

All requests to create and modify resources are evaluated against each **LimitRange** object in the project. If the resource violates any of the enumerated constraints, the resource is rejected.

The following shows a limit range object for all components: pod, container, image, image stream, or PVC. You can configure limits for any or all of these components in the same object. You create a different limit range object for each project where you want to control resources.

Sample limit range object for a container

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
  - type: "Container"
    max:
      cpu: "2"
      memory: "1Gi"
    min:
      cpu: "100m"
      memory: "4Mi"
    default:
      cpu: "300m"
      memory: "200Mi"
    defaultRequest:
      cpu: "200m"
      memory: "100Mi"
    maxLimitRequestRatio:
      cpu: "10"
```

8.3.2. About component limits

Review the following examples to learn the limit range parameters for each component for when you create or edit a **LimitRange** object.

The examples are broken out for clarity. You can create a single **LimitRange** object for any or all components as necessary.

Container limits

A limit range allows you to specify the minimum and maximum CPU and memory that each container in a pod can request for a specific project. If a container is created in the project, the container CPU and memory requests in the **Pod** spec must comply with the values set in the **LimitRange** object. If not, the pod does not get created. The following requirements must hold true:

- The container CPU or memory request and limit must be greater than or equal to the **min** resource constraint for containers that are specified in the **LimitRange** object.
- The container CPU or memory request and limit must be less than or equal to the **max** resource constraint for containers that are specified in the **LimitRange** object.

If the **LimitRange** object defines a **max** CPU, you do not need to define a CPU **request** value in the **Pod** spec. But you must specify a CPU **limit** value that satisfies the maximum CPU constraint specified in the limit range. The following requirements must hold true:

- The ratio of the container limits to requests must be less than or equal to the **maxLimitRequestRatio** value for containers that is specified in the **LimitRange** object. If the **LimitRange** object defines a **maxLimitRequestRatio** constraint, any new containers must have both a **request** and a **limit** value. OpenShift Container Platform calculates the limit-to-request ratio by dividing the **limit** by the **request**. This value should be a non-negative integer greater than 1.

For example, if a container has **cpu: 500** in the **limit** value, and **cpu: 100** in the **request** value, the limit-to-request ratio for **cpu** is **5**. This ratio must be less than or equal to the **maxLimitRequestRatio**.

If the **Pod** spec does not specify a container resource memory or limit, the **default** or **defaultRequest** CPU and memory values for containers specified in the limit range object are assigned to the container.

Container **LimitRange** object definition

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: "Container"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "100m"
        memory: "4Mi"
      default:
        cpu: "300m"
        memory: "200Mi"
      defaultRequest:
        cpu: "200m"
        memory: "100Mi"
      maxLimitRequestRatio:
        cpu: "10"
```

where:

metadata.name

Specifies the name of the limit range object.

spec.limit.max.cpu

Specifies the maximum amount of CPU that a single container in a pod can request.

spec.limit.max.memory

Specifies the maximum amount of memory that a single container in a pod can request.

spec.limit.min.cpu

Specifies the minimum amount of CPU that a single container in a pod can request.

spec.limit.min.memory

Specifies the minimum amount of memory that a single container in a pod can request.

spec.limit.default.cpu

Specifies the default amount of CPU that a container can use if not specified in the **Pod** spec.

spec.limit.default.memory

Specifies the default amount of memory that a container can use if not specified in the **Pod** spec.

spec.limit.defaultRequest.cpu

Specifies the default amount of CPU that a container can request if not specified in the **Pod** spec.

spec.limit.defaultRequest.memory

Specifies the default amount of memory that a container can request if not specified in the **Pod** spec.

spec.limit.maxLimitRequestRatio.cpu

Specifies the maximum limit-to-request ratio for a container.

Pod limits

A limit range allows you to specify the minimum and maximum CPU and memory limits for all containers across a pod in a given project. To create a container in the project, the container CPU and memory requests in the **Pod** spec must comply with the values set in the **LimitRange** object. If not, the pod does not get created.

If the **Pod** spec does not specify a container resource memory or limit, the **default** or **defaultRequest** CPU and memory values for containers specified in the limit range object are assigned to the container.

Across all containers in a pod, the following requirements must hold true:

- The container CPU or memory request and limit must be greater than or equal to the **min** resource constraints for pods that are specified in the **LimitRange** object.
- The container CPU or memory request and limit must be less than or equal to the **max** resource constraints for pods that are specified in the **LimitRange** object.
- The ratio of the container limits to requests must be less than or equal to the **maxLimitRequestRatio** constraint specified in the **LimitRange** object.

Pod LimitRange object definition

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "200m"
        memory: "6Mi"
      maxLimitRequestRatio:
        cpu: "10"
```

where:

metadata.name

Specifies the name of the limit range object.

spec.limit.max.cpu

Specifies the maximum amount of CPU that a pod can request across all containers.

spec.limit.max.memory

Specifies the maximum amount of memory that a pod can request across all containers.

spec.limit.min.cpu

Specifies the minimum amount of CPU that a pod can request across all containers.

spec.limit.min.memory

Specifies the minimum amount of memory that a pod can request across all containers.

spec.limit.maxLimitRequestRatio.cpu

Specifies the maximum limit-to-request ratio for a container.

Image limits

A limit range allows you to specify the maximum size of an image that can be pushed to an OpenShift image registry.

When pushing images to an OpenShift image registry, the following requirement must hold true:

- The size of the image must be less than or equal to the **max** size for images that is specified in the **LimitRange** object.

Image LimitRange object definition

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: openshift.io/Image
      max:
        storage: 1Gi
```

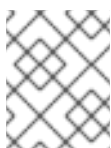
where:

metadata.name

Specifies the name of the limit range object.

spec.limit.max.storage

Specifies the maximum size of an image that can be pushed to an OpenShift image registry.



NOTE

To prevent blobs that exceed the limit from being uploaded to the registry, the registry must be configured to enforce quotas.



WARNING

The image size is not always available in the manifest of an uploaded image. This is especially the case for images built with Docker 1.10 or higher and pushed to a v2 registry. If such an image is pulled with an older Docker daemon, the image manifest is converted by the registry to schema v1 lacking all the size information. No storage limit set on images prevent it from being uploaded.

[The issue is being addressed.](#)

Image stream limits

A limit range allows you to specify limits for image streams.

For each image stream, the following requirements must hold true:

- The number of image tags in an **ImageStream** specification must be less than or equal to the **openshift.io/image-tags** constraint in the **LimitRange** object.
- The number of unique references to images in an **ImageStream** specification must be less than or equal to the **openshift.io/images** constraint in the limit range object.

Imagestream LimitRange object definition

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: openshift.io/ImageStream
      max:
        openshift.io/image-tags: 20
        openshift.io/images: 30
```

where

metadata.name

Specifies the name of the **LimitRange** object.

spec.limit.max.openshift.io/image-tags

Specifies the maximum number of unique image tags in the **imagestream.spec.tags** parameter in imagestream spec.

spec.limit.max.openshift.io/images

Specifies the maximum number of unique image references in the **imagestream.status.tags** parameter in the **imagestream** spec.

The **openshift.io/image-tags** resource represents unique image references. Possible references are an **ImageStreamTag**, an **ImageStreamImage** and a **DockerImage**. Tags can be created using the **oc tag** and **oc import-image** commands. No distinction is made between internal and external

references. However, each unique reference tagged in an **ImageStream** specification is counted just once. It does not restrict pushes to an internal container image registry in any way, but is useful for tag restriction.

The **openshift.io/images** resource represents unique image names recorded in image stream status. It allows for restriction of a specific number of images that can be pushed to the OpenShift image registry. Internal and external references are not distinguished.

Persistent volume claim limits

A limit range allows you to restrict the storage requested in a persistent volume claim (PVC). Across all persistent volume claims in a project, the following requirements must hold true:

- The resource request in a persistent volume claim (PVC) must be greater than or equal the **min** constraint for PVCs that is specified in the **LimitRange** object.
- The resource request in a persistent volume claim (PVC) must be less than or equal the **max** constraint for PVCs that is specified in the **LimitRange** object.

PVC LimitRange object definition

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: "PersistentVolumeClaim"
      min:
        storage: "2Gi"
      max:
        storage: "50Gi"
```

where:

metadata.name

Specifies the name of the **LimitRange** object.

spec.limit.min.storage

Specifies the minimum amount of storage that can be requested in a persistent volume claim.

spec.limit.max.storage

Specifies the maximum amount of storage that can be requested in a persistent volume claim.

8.3.3. Creating a Limit Range

You can define **LimitRange** objects to set specific resource limits for a pod, container, image, image stream, or persistent volume claim (PVC) in a specific project. A limit range allows you to restrict resource consumption in that project.

Procedure

1. Create a **LimitRange** object with your required specifications:

```

apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "200m"
        memory: "6Mi"
    - type: "Container"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "100m"
        memory: "4Mi"
      default:
        cpu: "300m"
        memory: "200Mi"
      defaultRequest:
        cpu: "200m"
        memory: "100Mi"
      maxLimitRequestRatio:
        cpu: "10"
    - type: openshift.io/Image
      max:
        storage: 1Gi
    - type: openshift.io/ImageStream
      max:
        openshift.io/image-tags: 20
        openshift.io/images: 30
    - type: "PersistentVolumeClaim"
      min:
        storage: "2Gi"
      max:
        storage: "50Gi"

```

where:

metadata.name

Specifies a name for the **LimitRange** object.

spec.limit.type.Pod

Specifies limits for a pod, specify the minimum and maximum CPU and memory requests as needed.

spec.limit.type.Container

Specifies limits for a container, specify the minimum and maximum CPU and memory requests as needed.

spec.limit.type.default

For a container, specifies the default amount of CPU or memory that a container can use, if not specified in the **Pod** spec. This parameter is optional.

spec.limit.type.defaultRequest

For a container, specifies the default amount of CPU or memory that a container can request, if not specified in the **Pod** spec. This parameter is optional.

spec.limit.type.maxLimitRequestRatio

For a container, specifies the maximum limit-to-request ratio that can be specified in the **Pod** spec. This parameter is optional.

spec.limit.type.openshift.io/Image

Specifies limits for an image object. Set the maximum size of an image that can be pushed to an OpenShift image registry.

spec.limit.type.openshift.io/ImageStream

Specifies limits for an image stream. Set the maximum number of image tags and references that can be in the **ImageStream** object file, as needed.

spec.limit.type.openshift.io/PersistentVolumeClaim

Specifies limits for a persistent volume claim. Set the minimum and maximum amount of storage that can be requested.

2. Create the object:

```
$ oc create -f <limit_range_file> -n <project>
```

where:

<limit_range_file>

Specifies the name of the YAML file you created.

<project>

Specifies the project where you want the limits to apply.

8.3.4. Viewing a limit

You can view the limits defined in a project by navigating in the web console to the project's **Quota** page. This allows you to see details about each of the limit ranges in a project.

You can also use the CLI to view limit range details:

Procedure

1. Get the list of **LimitRange** objects defined in the project. For example, for a project called **demoproject**:

```
$ oc get limits -n demoproject
```

```
NAME          CREATED AT
resource-limits 2020-07-15T17:14:23Z
```

2. Describe the **LimitRange** object you are interested in, for example the **resource-limits** limit range:

```
$ oc describe limits resource-limits -n demoproject
```

Name:	resource-limits						
Namespace:	demoproject						
Type	Resource	Min	Max	Default	Request	Default Limit	Max
Limit/Request Ratio							
Pod	cpu	200m	2	-	-	-	
Pod	memory	6Mi	1Gi	-	-	-	
Container	cpu	100m	2	200m	300m	10	
Container	memory	4Mi	1Gi	100Mi	200Mi	-	
openshift.io/Image	storage	-	1Gi	-	-	-	
openshift.io/ImageStream	openshift.io/image	-	12	-	-	-	
openshift.io/ImageStream	openshift.io/image-tags	-	10	-	-	-	
PersistentVolumeClaim	storage	-	50Gi	-	-	-	

8.3.5. Deleting a Limit Range

You can remove any active **LimitRange** object so that it no longer enforces the limits in a project.

Procedure

- Run the following command:

```
$ oc delete limits <limit_name>
```

8.4. CONFIGURING CLUSTER MEMORY TO MEET CONTAINER MEMORY AND RISK REQUIREMENTS

As a cluster administrator, you can manage application memory usage to help your clusters operate more efficiently.

You can perform any of the following tasks to manage application memory:

- Determine the memory and risk requirements of a containerized application component and configuring the container memory parameters to suit those requirements.
- Configure containerized application runtimes (for example, OpenJDK) to adhere optimally to the configured container memory parameters.
- Diagnose and resolve memory-related error conditions associated with running in a container.

8.4.1. Understanding how to manage application memory

You can review the following concepts to learn how OpenShift Container Platform manages compute resources so that you can learn how to keep your cluster running efficiently.

For each kind of resource (memory, CPU, storage), OpenShift Container Platform allows optional **request** and **limit** values to be placed on each container in a pod.

Note the following information about memory requests and memory limits:

- Memory request**

- The memory request value, if specified, influences the OpenShift Container Platform scheduler. The scheduler considers the memory request when scheduling a container to a node, then fences off the requested memory on the chosen node for the use of the container.
 - If a node's memory is exhausted, OpenShift Container Platform prioritizes evicting its containers whose memory usage most exceeds their memory request. In serious cases of memory exhaustion, the node OOM killer might select and kill a process in a container based on a similar metric.
 - The cluster administrator can assign quota or assign default values for the memory request value.
 - The cluster administrator can override the memory request values that a developer specifies, to manage cluster overcommit.
- **Memory limit**
 - The memory limit value, if specified, provides a hard limit on the memory that can be allocated across all the processes in a container.
 - If the memory allocated by all of the processes in a container exceeds the memory limit, the node Out of Memory (OOM) killer immediately selects and kills a process in the container.
 - If both memory request and limit are specified, the memory limit value must be greater than or equal to the memory request.
 - The cluster administrator can assign quota or assign default values for the memory limit value.
 - The minimum memory limit is 12 MB. If a container fails to start due to a **Cannot allocate memory** pod event, the memory limit is too low. Either increase or remove the memory limit. Removing the limit allows pods to consume unbounded node resources.

The steps for sizing application memory on OpenShift Container Platform are as follows:

1. **Determine expected container memory usage**
Determine expected mean and peak container memory usage. For example, you could perform separate load testing. Remember to consider all the processes that could potentially run in parallel in the container, such as any ancillary scripts that might be spawned by the main application.
2. **Determine risk appetite**
Determine risk appetite for eviction. If the risk appetite is low, the container should request memory according to the expected peak usage plus a percentage safety margin. If the risk appetite is higher, it might be more appropriate to request memory according to the expected mean usage.
3. **Set container memory request**
Set the container memory request based on the above. The request should represent the application memory usage as accurately as possible. If the request is too high, cluster and quota usage will be inefficient. If the request is too low, the chances of application eviction increase.
4. **Set container memory limit, if required**
Set the container memory limit, if required. Setting a limit has the effect of immediately killing a container process if the combined memory usage of all processes in the container exceeds the limit. Setting a limit might make unanticipated excess memory usage obvious early (*fail fast*).

However, setting a limit also terminates processes abruptly.

Note that some OpenShift Container Platform clusters might require a limit value to be set; some might override the request based on the limit; and some application images rely on a limit value being set as this is easier to detect than a request value.

If the memory limit is set, it should not be set to less than the expected peak container memory usage plus a percentage safety margin.

5. Ensure applications are tuned

Ensure your applications are tuned with respect to configured request and limit values, if appropriate. This step is particularly relevant to applications which pool memory, such as the JVM. The rest of this page discusses this.

8.4.2. Understanding OpenJDK settings for OpenShift Container Platform

You can review the following concepts to learn about how to deploy OpenJDK applications in your cluster effectively.

The default OpenJDK settings do not work well with containerized environments. As a result, some additional Java memory settings must always be provided whenever running the OpenJDK in a container.

The JVM memory layout is complex, version dependent, and describing it in detail is beyond the scope of this documentation. However, as a starting point for running OpenJDK in a container, at least the following three memory-related tasks are key:

Overriding the JVM maximum heap size

OpenJDK defaults to using a maximum of 25% of available memory (recognizing any container memory limits in place) for *heap* memory. This default value is conservative, and, in a properly-configured container environment, would result in 75% of the memory assigned to a container being mostly unused. A much higher percentage for the JVM to use for heap memory, such as 80%, is more suitable in a container context where memory limits are imposed on the container level. Most of the Red Hat containers include a startup script that replaces the OpenJDK default by setting updated values when the JVM launches.

For example, the Red Hat build of OpenJDK containers have a default value of 80%. This value can be set to a different percentage by defining the **JAVA_MAX_RAM_RATIO** environment variable.

For other OpenJDK deployments, the default value of 25% can be changed using the following command:

Example

```
$ java -XX:MaxRAMPercentage=80.0
```

Encouraging the JVM to release unused memory to the operating system, if appropriate

By default, the OpenJDK does not aggressively return unused memory to the operating system. This could be appropriate for many containerized Java workloads, but notable exceptions include workloads where additional active processes co-exist with a JVM within a container, whether those additional processes are native, additional JVMs, or a combination of the two.

Java-based agents can use the following JVM arguments to encourage the JVM to release unused memory to the operating system:

```
-XX:+UseParallelGC
-XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4
-XX:AdaptiveSizePolicyWeight=90
```

These arguments are intended to return heap memory to the operating system whenever allocated memory exceeds 110% of in-use memory (**-XX:MaxHeapFreeRatio**), spending up to 20% of CPU time in the garbage collector (**-XX:GCTimeRatio**). At no time will the application heap allocation be less than the initial heap allocation (overridden by **-XX:InitialHeapSize** / **-Xms**). Detailed additional information is available [Tuning Java's footprint in OpenShift \(Part 1\)](#) , [Tuning Java's footprint in OpenShift \(Part 2\)](#), and at [OpenJDK and Containers](#).

Ensuring all JVM processes within a container are appropriately configured

In the case that multiple JVMs run in the same container, it is essential to ensure that they are all configured appropriately. For many workloads it will be necessary to grant each JVM a percentage memory budget, leaving a perhaps substantial additional safety margin.

Many Java tools use different environment variables (**JAVA_OPTS**, **GRADLE_OPTS**, and so on) to configure their JVMs and it can be challenging to ensure that the right settings are being passed to the right JVM.

The **JAVA_TOOL_OPTIONS** environment variable is always respected by the OpenJDK, and values specified in **JAVA_TOOL_OPTIONS** will be overridden by other options specified on the JVM command line. By default, to ensure that these options are used by default for all JVM workloads run in the Java-based agent image, the OpenShift Container Platform Jenkins Maven agent image sets the following variable:

```
JAVA_TOOL_OPTIONS="-Dsun.zip.disableMemoryMapping=true"
```

This does not guarantee that additional options are not required, but is intended to be a helpful starting point. Optimally tuning JVM workloads for running in a container is beyond the scope of this documentation, and may involve setting multiple additional JVM options.

8.4.3. Finding the memory request and limit from within a pod

You can configure your container to use the Downward API to dynamically discover its memory request and limit from within a pod. This allows your applications to better manage these resources without needing to use the API server.

Procedure

- Configure the pod to add the **MEMORY_REQUEST** and **MEMORY_LIMIT** stanzas:
 - a. Create a YAML file similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  securityContext:
    runAsNonRoot: false
  seccompProfile:
    type: RuntimeDefault
  containers:
```

```

- name: test
  image: fedora:latest
  command:
  - sleep
  - "3600"
  env:
  - name: MEMORY_REQUEST
    valueFrom:
      resourceFieldRef:
        containerName: test
        resource: requests.memory
  - name: MEMORY_LIMIT
    valueFrom:
      resourceFieldRef:
        containerName: test
        resource: limits.memory
  resources:
    requests:
      memory: 384Mi
    limits:
      memory: 512Mi
  securityContext:
    allowPrivilegeEscalation: false
  capabilities:
    drop: [ALL]

```

where:

spec.consinters.env.name.MEMORY_REQUEST

This stanza discovers the application memory request value.

spec.consinters.env.name.MEMORY_LIMIT

This stanza discovers the application memory limit value.

- b. Create the pod by running the following command:

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

Verification

1. Access the pod using a remote shell:

```
$ oc rsh test
```

2. Check that the requested values were applied:

```
$ env | grep MEMORY | sort
```

Example output

```

MEMORY_LIMIT=536870912
MEMORY_REQUEST=402653184

```



NOTE

The memory limit value can also be read from inside the container by the `/sys/fs/cgroup/memory/memory.limit_in_bytes` file.

8.4.4. Understanding OOM kill policy

OpenShift Container Platform can kill a process in a container if the total memory usage of all the processes in the container exceeds the memory limit, or in serious cases of node memory exhaustion.

If a process is Out of Memory (OOM) killed, the container could exit immediately. If the container PID 1 process receives the **SIGKILL**, the container does exit immediately. Otherwise, the container behavior is dependent on the behavior of the other processes.

For example, a container process exited with code 137, indicating it received a SIGKILL signal.

If the container does not exit immediately, use the following steps to detect if an OOM kill occurred.

Procedure

1. Access the pod using a remote shell:

```
# oc rsh <pod name>
```

2. Run the following command to see the current OOM kill count in `/sys/fs/cgroup/memory/memory.oom_control`:

```
$ grep '^oom_kill' /sys/fs/cgroup/memory/memory.oom_control
```

Example output

```
oom_kill 0
```

3. Run the following command to provoke an OOM kill:

```
$ sed -e " </dev/zero
```

Example output

```
Killed
```

4. Run the following command to see that the OOM kill counter in `/sys/fs/cgroup/memory/memory.oom_control` incremented:

```
$ grep '^oom_kill' /sys/fs/cgroup/memory/memory.oom_control
```

Example output

```
oom_kill 1
```

If one or more processes in a pod are OOM killed, when the pod subsequently exits, whether immediately or not, it will have phase **Failed** and reason **OOMKilled**. An OOM-killed pod might

be restarted depending on the value of **restartPolicy**. If not restarted, controllers such as the replication controller will notice the pod's failed status and create a new pod to replace the old one.

Use the following command to get the pod status:

```
$ oc get pod test
```

Example output

```
NAME    READY   STATUS    RESTARTS   AGE
test    0/1     OOMKilled 0           1m
```

- If the pod has not restarted, run the following command to view the pod:

```
$ oc get pod test -o yaml
```

Example output

```
apiVersion: v1
kind: Pod
metadata:
  name: test
# ...
status:
  containerStatuses:
  - name: test
    ready: false
    restartCount: 0
    state:
      terminated:
        exitCode: 137
        reason: OOMKilled
  phase: Failed
```

- If restarted, run the following command to view the pod:

```
$ oc get pod test -o yaml
```

Example output

```
apiVersion: v1
kind: Pod
metadata:
  name: test
# ...
status:
  containerStatuses:
  - name: test
    ready: true
    restartCount: 1
    lastState:
      terminated:
```



```

    exitCode: 137
    reason: OOMKilled
    state:
      running:
    phase: Running

```

8.4.5. Understanding pod eviction

You can review the following concepts to learn the OpenShift Container Platform pod eviction policy.

OpenShift Container Platform can evict a pod from its node when the node's memory is exhausted. Depending on the extent of memory exhaustion, the eviction might or might not be graceful. Graceful eviction implies the main process (PID 1) of each container receiving a SIGTERM signal, then some time later a SIGKILL signal if the process has not exited already. Non-graceful eviction implies the main process of each container immediately receiving a SIGKILL signal.

An evicted pod has phase **Failed** and reason **Evicted**. It is not restarted, regardless of the value of **restartPolicy**. However, controllers such as the replication controller will notice the pod's failed status and create a new pod to replace the old one.

```
$ oc get pod test
```

Example output

```

NAME    READY   STATUS    RESTARTS   AGE
test    0/1     Evicted   0           1m

```

```
$ oc get pod test -o yaml
```

Example output

```

apiVersion: v1
kind: Pod
metadata:
  name: test
...
status:
  message: 'Pod The node was low on resource: [MemoryPressure].'
  phase: Failed
  reason: Evicted

```

Additional resources

- [Understanding compute resources and containers](#)

8.5. CONFIGURING YOUR CLUSTER TO PLACE PODS ON OVERCOMMITTED NODES

OpenShift Container Platform administrators can control the level of overcommit and manage container density on developer containers by using the ClusterResourceOverride Operator.



NOTE

In OpenShift Container Platform, you must enable cluster-level overcommit. Node overcommitment is enabled by default.

In an *overcommitted* state, the sum of the container compute resource requests and limits exceeds the resources available on the system. For example, you might want to use overcommitment in development environments where a trade-off of guaranteed performance for capacity is acceptable.

Containers can specify compute resource requests and limits. Requests are used for scheduling your container and provide a minimum service guarantee. Limits constrain the amount of compute resource that can be consumed on your node.

The scheduler attempts to optimize the compute resource use across all nodes in your cluster. It places pods onto specific nodes, taking the pods' compute resource requests and nodes' available capacity into consideration.

8.5.1. 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.

8.5.2. 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
      limitCPUToMemoryPercent: 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.limitCPUToMemoryPercent

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.

8.5.2.1. 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.limitCPUMemoryPercent

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:
    kubectrl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride","met
adata":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":{"spec":
{"cpuRequestToLimitPercent":25,"limitCPUMemoryPercent":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
      limitCPUMemoryPercent: 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.

8.5.2.2. 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-namespace.yaml
```

2. Create an Operator group:
 - a. Create an **OperatorGroup** object YAML file (for example, **cro-og.yaml**) for the Cluster Resource Override Operator:

```

apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: clusterresourceoverride-operator
  namespace: clusterresourceoverride-operator
spec:
  targetNamespaces:
    - clusterresourceoverride-operator

```

- b. Create the Operator Group:

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

For example:

```
$ oc create -f cro-og.yaml
```

3. Create a subscription:

- a. Create a **Subscription** object YAML file (for example, cro-sub.yaml) for the Cluster Resource Override Operator:

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: clusterresourceoverride
  namespace: clusterresourceoverride-operator
spec:
  channel: "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
      limitCPUMemoryPercent: 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.limitCPUMemoryPercent

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}}}}
```

```

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.

8.5.2.3. 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.

8.5.2.4. Moving the Cluster Resource Override Operator pods

By default, the Cluster Resource Override Operator installation process creates an Operator pod and two Cluster Resource Override pods on nodes in the **clusterresourceoverride-operator** namespace. You can move these pods to other nodes, such as infrastructure nodes, as needed.

You can create and use infrastructure nodes to host only infrastructure components, such as the default router, the integrated container image registry, and the components for cluster metrics and monitoring. These infrastructure nodes are not counted toward the total number of subscriptions that are required

to run the environment. For more information about infrastructure nodes, see "Creating infrastructure machine sets".

The following examples shows the Cluster Resource Override pods are deployed to control plane nodes and the Cluster Resource Override Operator pod is deployed to a worker node.

Example Cluster Resource Override pods

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
NOMINATED NODE READINESS GATES						
clusterresourceoverride-786b8c898c-9wrdq	1/1	Running	0	23s	10.128.2.32	ip-10-0-14-183.us-west-2.compute.internal
clusterresourceoverride-786b8c898c-vn2lf	1/1	Running	0	26s	10.130.2.10	ip-10-0-20-140.us-west-2.compute.internal
clusterresourceoverride-operator-6b8b8b656b-lvr62	1/1	Running	0	56m	10.131.0.33	ip-10-0-2-39.us-west-2.compute.internal

Example node list

NAME	STATUS	ROLES	AGE	VERSION
ip-10-0-14-183.us-west-2.compute.internal	Ready	control-plane,master	65m	v1.33.4
ip-10-0-2-39.us-west-2.compute.internal	Ready	worker	58m	v1.33.4
ip-10-0-20-140.us-west-2.compute.internal	Ready	control-plane,master	65m	v1.33.4
ip-10-0-23-244.us-west-2.compute.internal	Ready	infra	55m	v1.33.4
ip-10-0-77-153.us-west-2.compute.internal	Ready	control-plane,master	65m	v1.33.4
ip-10-0-99-108.us-west-2.compute.internal	Ready	worker	24m	v1.33.4
ip-10-0-24-233.us-west-2.compute.internal	Ready	infra	55m	v1.33.4
ip-10-0-88-109.us-west-2.compute.internal	Ready	worker	24m	v1.33.4
ip-10-0-67-453.us-west-2.compute.internal	Ready	infra	55m	v1.33.4

Procedure

1. Move the Cluster Resource Override Operator pod by adding a node selector to the **Subscription** custom resource (CR) for the Cluster Resource Override Operator.

- a. Edit the CR:

```
$ oc edit -n clusterresourceoverride-operator subscriptions.operators.coreos.com
clusterresourceoverride
```

- b. Add a node selector to match the node role label on the node where you want to install the Cluster Resource Override Operator pod:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: clusterresourceoverride
  namespace: clusterresourceoverride-operator
# ...
spec:
  config:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
# ...
```

where

spec.config.nodeSelector

Specifies the role of the node where you want to deploy the Cluster Resource Override Operator pod.

NOTE

If the infra node uses taints, you need to add a toleration to the **Subscription** CR.

For example:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: clusterresourceoverride
  namespace: clusterresourceoverride-operator
# ...
spec:
  config:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
      - key: "node-role.kubernetes.io/infra"
        operator: "Exists"
        effect: "NoSchedule"
```

where:

spec.config.tolerations

Specifies a toleration for a taint on the infra node.

2. Move the Cluster Resource Override pods by adding a node selector to the **ClusterResourceOverride** custom resource (CR):

- a. Edit the CR:

```
$ oc edit ClusterResourceOverride cluster -n clusterresourceoverride-operator
```

- b. Add a node selector to match the node role label on the infra node:

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster
  resourceVersion: "37952"
spec:
  podResourceOverride:
    spec:
      cpuRequestToLimitPercent: 25
      limitCPUToMemoryPercent: 200
      memoryRequestToLimitPercent: 50
  deploymentOverrides:
```

```

replicas: 1 1
nodeSelector:
  node-role.kubernetes.io/infra: "" 2
# ...

```

where

spec.deploymentOverrides.replicas

Specifies the number of Cluster Resource Override pods to deploy. The default is **2**. Only one pod is allowed per node. This parameter is optional.

spec.deploymentOverrides.nodeSelector

Specifies the role of the node where you want to deploy the Cluster Resource Override pods. This parameter is optional.

NOTE

If the infra node uses taints, you need to add a toleration to the **ClusterResourceOverride** CR.

For example:

```

apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster
# ...
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50
      cpuRequestToLimitPercent: 25
      limitCPUToMemoryPercent: 200
  deploymentOverrides:
    replicas: 3
    nodeSelector:
      node-role.kubernetes.io/worker: ""
    tolerations: 1
    - key: "key"
      operator: "Equal"
      value: "value"
      effect: "NoSchedule"

```

where:

+

spec.config.tolerations

Specifies a toleration for a taint on the infra node.

Verification

- You can verify that the pods have moved by using the following command:

```
$ oc get pods -n clusterresourceoverride-operator -o wide
```

The Cluster Resource Override pods are now deployed to the infra nodes.

Example output

NAME	READY	STATUS	RESTARTS	AGE	IP
NODE	NOMINATED NODE	READINESS GATES			
clusterresourceoverride-786b8c898c-9wrdq	1/1	Running	0	23s	10.127.2.25
ip-10-0-23-244.us-west-2.compute.internal	<none>	<none>			
clusterresourceoverride-786b8c898c-vn2lf	1/1	Running	0	26s	10.128.0.80
ip-10-0-24-233.us-west-2.compute.internal	<none>	<none>			
clusterresourceoverride-operator-6b8b8b656b-lvr62	1/1	Running	0	56m	
10.129.0.71 ip-10-0-67-453.us-west-2.compute.internal	<none>	<none>			

8.5.3. 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.

8.5.3.1. 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.

8.5.3.2. 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 8.17. 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 .
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.

8.5.3.2.1. 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.

8.5.3.3. 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.

8.5.3.4. 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

8.5.3.5. 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
```

8.5.3.6. 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.

8.5.3.7. 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
```

8.5.4. 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.

8.5.4.1. 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.

8.5.5. Additional resources

- [Moving the Cluster Resource Override Operator pods](#)
- [Creating infrastructure machine sets](#)
- [Setting deployment 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](#)
- [Allocating resources for nodes](#)

8.6. ENABLING FEATURES USING FEATURE GATES

As an administrator, you can use feature gates to enable features that are not part of the default set of features so that you can use these non-default features in your cluster.

8.6.1. 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.

8.6.2. Enabling feature sets at installation

You can enable feature sets for all nodes in the cluster by editing the **install-config.yaml** file before you deploy the cluster. This allows you to use non-default features in your cluster.

Prerequisites

- You have an **install-config.yaml** file.

Procedure

1. Use the **featureSet** parameter to specify the name of the feature set you want to enable, such as **TechPreviewNoUpgrade**:



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 install-config.yaml file with an enabled feature set

```
compute:
- hyperthreading: Enabled
  name: worker
  platform:
    aws:
      rootVolume:
        iops: 2000
        size: 500
        type: io1
      metadataService:
        authentication: Optional
```



```

type: c5.4xlarge
zones:
  - us-west-2c
replicas: 3
featureSet: TechPreviewNoUpgrade

```

2. Save the file and reference it when using the installation program to deploy the cluster.

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.

8.6.3. 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.

8.6.4. 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.

8.6.5. Additional resources

- [Shared Resources CSI Driver and Build CSI Volumes in OpenShift Builds](#)
- [CSI inline ephemeral volumes](#)
- [Swap memory on nodes](#)
- [Managing machines with the Cluster API](#)
- [Disabling the Insights Operator gather operations](#)
- [Enabling the Insights Operator gather operations](#)
- [Running an Insights Operator gather operation](#)
- [Managing the default storage class](#)
- [Pod security admission enforcement](#)

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

Review the following information to learn about *worker latency profiles*, which adjust the frequency that the Kubelet and the Kubernetes Controller Manager wait for status updates before taking action if a pod is unreachable.

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.

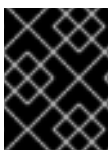
These worker latency profiles contain three sets of parameters that are predefined with carefully tuned values to control the reaction of the cluster to increased latency. There is no need to experimentally find the best values manually.

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

8.7.1. 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

Profile	Component	Parameter	Value
	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.

8.7.2. 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.

CHAPTER 9. REMOTE WORKER NODES ON THE NETWORK EDGE

9.1. USING REMOTE WORKER NODES AT THE NETWORK EDGE

You can configure OpenShift Container Platform clusters with nodes located at your network edge. In this topic, they are called *remote worker nodes*. A typical cluster with remote worker nodes combines on-premise master and worker nodes with worker nodes in other locations that connect to the cluster. This topic is intended to provide guidance on best practices for using remote worker nodes and does not contain specific configuration details.

There are multiple use cases across different industries, such as retail, manufacturing, and government, for using a deployment pattern with remote worker nodes. For example, you can separate and isolate your projects and workloads by combining the remote worker nodes into [Kubernetes zones](#).

However, having remote worker nodes can introduce higher latency, intermittent loss of network connectivity, and other issues. Among the challenges in a cluster with remote worker node are:

- **Network separation:** The OpenShift Container Platform control plane and the remote worker nodes must be able to communicate with each other. Because of the distance between the control plane and the remote worker nodes, network issues could prevent this communication. See [Network separation with remote worker nodes](#) for information on how OpenShift Container Platform responds to network separation and for methods to diminish the impact to your cluster.
- **Power outage:** Because the control plane and remote worker nodes are in separate locations, a power outage at the remote location or at any point between the two can negatively impact your cluster. See [Power loss on remote worker nodes](#) for information on how OpenShift Container Platform responds to a node losing power and for methods to diminish the impact to your cluster.
- **Latency spikes or temporary reduction in throughput** As with any network, any changes in network conditions between your cluster and the remote worker nodes can negatively impact your cluster. OpenShift Container Platform offers multiple *worker latency profiles* that let you control the reaction of the cluster to latency issues.

Note the following limitations when planning a cluster with remote worker nodes:

- OpenShift Container Platform does not support remote worker nodes that use a different cloud provider than the on-premise cluster uses.
- Moving workloads from one Kubernetes zone to a different Kubernetes zone can be problematic due to system and environment issues, such as a specific type of memory not being available in a different zone.
- Proxies and firewalls can present additional limitations that are beyond the scope of this document. See the relevant OpenShift Container Platform documentation for how to address such limitations, such as [Configuring your firewall](#).
- You are responsible for configuring and maintaining L2/L3-level network connectivity between the control plane and the network-edge nodes.

9.1.1. Adding remote worker nodes

Adding remote worker nodes to a cluster involves some additional considerations.

- You must ensure that a route or a default gateway is in place to route traffic between the control plane and every remote worker node.
- You must place the Ingress VIP on the control plane.
- Adding remote worker nodes with user-provisioned infrastructure is identical to adding other worker nodes.
- To add remote worker nodes to an installer-provisioned cluster at install time, specify the subnet for each worker node in the **install-config.yaml** file before installation. There are no additional settings required for the DHCP server. You must use virtual media, because the remote worker nodes will not have access to the local provisioning network.
- To add remote worker nodes to an installer-provisioned cluster deployed with a provisioning network, ensure that **virtualMediaViaExternalNetwork** flag is set to **true** in the **install-config.yaml** file so that it will add the nodes using virtual media. Remote worker nodes will not have access to the local provisioning network. They must be deployed with virtual media rather than PXE. Additionally, specify each subnet for each group of remote worker nodes and the control plane nodes in the DHCP server.

Additional resources

- [Establishing communications between subnets](#)
- [Configuring host network interfaces for subnets](#)
- [Configuring network components to run on the control plane](#)

9.1.2. Network separation with remote worker nodes

All nodes send heartbeats to the Kubernetes Controller Manager Operator (kube controller) in the OpenShift Container Platform cluster every 10 seconds. If the cluster does not receive heartbeats from a node, OpenShift Container Platform responds using several default mechanisms.

OpenShift Container Platform is designed to be resilient to network partitions and other disruptions. You can mitigate some of the more common disruptions, such as interruptions from software upgrades, network splits, and routing issues. Mitigation strategies include ensuring that pods on remote worker nodes request the correct amount of CPU and memory resources, configuring an appropriate replication policy, using redundancy across zones, and using Pod Disruption Budgets on workloads.

If the kube controller loses contact with a node after a configured period, the node controller on the control plane updates the node health to **Unhealthy** and marks the node **Ready** condition as **Unknown**. In response, the scheduler stops scheduling pods to that node. The on-premise node controller adds a **node.kubernetes.io/unreachable** taint with a **NoExecute** effect to the node and schedules pods on the node for eviction after five minutes, by default.

If a workload controller, such as a **Deployment** object or **StatefulSet** object, is directing traffic to pods on the unhealthy node and other nodes can reach the cluster, OpenShift Container Platform routes the traffic away from the pods on the node. Nodes that cannot reach the cluster do not get updated with the new traffic routing. As a result, the workloads on those nodes might continue to attempt to reach the unhealthy node.

You can mitigate the effects of connection loss by:

- using daemon sets to create pods that tolerate the taints

- using static pods that automatically restart if a node goes down
- using Kubernetes zones to control pod eviction
- configuring pod tolerations to delay or avoid pod eviction
- configuring the kubelet to control the timing of when it marks nodes as unhealthy.

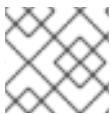
For more information on using these objects in a cluster with remote worker nodes, see [About remote worker node strategies](#).

9.1.3. Power loss on remote worker nodes

If a remote worker node loses power or restarts ungracefully, OpenShift Container Platform responds using several default mechanisms.

If the Kubernetes Controller Manager Operator (kube controller) loses contact with a node after a configured period, the control plane updates the node health to **Unhealthy** and marks the node **Ready** condition as **Unknown**. In response, the scheduler stops scheduling pods to that node. The on-premise node controller adds a **node.kubernetes.io/unreachable** taint with a **NoExecute** effect to the node and schedules pods on the node for eviction after five minutes, by default.

On the node, the pods must be restarted when the node recovers power and reconnects with the control plane.



NOTE

If you want the pods to restart immediately upon restart, use static pods.

After the node restarts, the kubelet also restarts and attempts to restart the pods that were scheduled on the node. If the connection to the control plane takes longer than the default five minutes, the control plane cannot update the node health and remove the **node.kubernetes.io/unreachable** taint. On the node, the kubelet terminates any running pods. When these conditions are cleared, the scheduler can start scheduling pods to that node.

You can mitigate the effects of power loss by:

- using daemon sets to create pods that tolerate the taints
- using static pods that automatically restart with a node
- configuring pods tolerations to delay or avoid pod eviction
- configuring the kubelet to control the timing of when the node controller marks nodes as unhealthy.

For more information on using these objects in a cluster with remote worker nodes, see [About remote worker node strategies](#).

9.1.4. Latency spikes or temporary reduction in throughput to remote workers

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.

Additional resources

- [Improving cluster stability in high latency environments using worker latency profiles](#)

9.1.5. Remote worker node strategies

If you use remote worker nodes, consider which objects to use to run your applications.

It is recommended to use daemon sets or static pods based on the behavior you want in the event of network issues or power loss. In addition, you can use Kubernetes zones and tolerations to control or avoid pod evictions if the control plane cannot reach remote worker nodes.

Daemon sets

Daemon sets are the best approach to managing pods on remote worker nodes for the following reasons:

- Daemon sets do not typically need rescheduling behavior. If a node disconnects from the cluster, pods on the node can continue to run. OpenShift Container Platform does not change the state of daemon set pods, and leaves the pods in the state they last reported. For example,

if a daemon set pod is in the **Running** state, when a node stops communicating, the pod keeps running and is assumed to be running by OpenShift Container Platform.

- Daemon set pods, by default, are created with **NoExecute** tolerations for the **node.kubernetes.io/unreachable** and **node.kubernetes.io/not-ready** taints with no **tolerationSeconds** value. These default values ensure that daemon set pods are never evicted if the control plane cannot reach a node. For example:

Tolerations added to daemon set pods by default

```
tolerations:
- key: node.kubernetes.io/not-ready
  operator: Exists
  effect: NoExecute
- key: node.kubernetes.io/unreachable
  operator: Exists
  effect: NoExecute
- key: node.kubernetes.io/disk-pressure
  operator: Exists
  effect: NoSchedule
- key: node.kubernetes.io/memory-pressure
  operator: Exists
  effect: NoSchedule
- key: node.kubernetes.io/pid-pressure
  operator: Exists
  effect: NoSchedule
- key: node.kubernetes.io/unschedulable
  operator: Exists
  effect: NoSchedule
```

- Daemon sets can use labels to ensure that a workload runs on a matching worker node.
- You can use an OpenShift Container Platform service endpoint to load balance daemon set pods.



NOTE

Daemon sets do not schedule pods after a reboot of the node if OpenShift Container Platform cannot reach the node.

Static pods

If you want pods restart if a node reboots, after a power loss for example, consider [static pods](#). The kubelet on a node automatically restarts static pods as node restarts.



NOTE

Static pods cannot use secrets and config maps.

Kubernetes zones

[Kubernetes zones](#) can slow down the rate or, in some cases, completely stop pod evictions.

When the control plane cannot reach a node, the node controller, by default, applies **node.kubernetes.io/unreachable** taints and evicts pods at a rate of 0.1 nodes per second. However, in a cluster that uses Kubernetes zones, pod eviction behavior is altered.

If a zone is fully disrupted, where all nodes in the zone have a **Ready** condition that is **False** or **Unknown**, the control plane does not apply the **node.kubernetes.io/unreachable** taint to the nodes in that zone.

For partially disrupted zones, where more than 55% of the nodes have a **False** or **Unknown** condition, the pod eviction rate is reduced to 0.01 nodes per second. Nodes in smaller clusters, with fewer than 50 nodes, are not tainted. Your cluster must have more than three zones for these behavior to take effect.

You assign a node to a specific zone by applying the **topology.kubernetes.io/region** label in the node specification.

Sample node labels for Kubernetes zones

```
kind: Node
apiVersion: v1
metadata:
  labels:
    topology.kubernetes.io/region=east
```

KubeletConfig objects

You can adjust the amount of time that the kubelet checks the state of each node.

To set the interval that affects the timing of when the on-premise node controller marks nodes with the **Unhealthy** or **Unreachable** condition, create a **KubeletConfig** object that contains the **node-status-update-frequency** and **node-status-report-frequency** parameters.

The kubelet on each node determines the node status as defined by the **node-status-update-frequency** setting and reports that status to the cluster based on the **node-status-report-frequency** setting. By default, the kubelet determines the pod status every 10 seconds and reports the status every minute. However, if the node state changes, the kubelet reports the change to the cluster immediately. OpenShift Container Platform uses the **node-status-report-frequency** setting only when the Node Lease feature gate is enabled, which is the default state in OpenShift Container Platform clusters. If the Node Lease feature gate is disabled, the node reports its status based on the **node-status-update-frequency** setting.

Example kubelet config

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: disable-cpu-units
spec:
  machineConfigPoolSelector:
    matchLabels:
      machineconfiguration.openshift.io/role: worker ❶
  kubeletConfig:
    node-status-update-frequency: ❷
    - "10s"
    node-status-report-frequency: ❸
    - "1m"
```

- 1 1 1 1 1 Specify the type of node type to which this **KubeletConfig** object applies using the label from the **MachineConfig** object.
- 2 2 2 Specify the frequency that the kubelet checks the status of a node associated with this **MachineConfig** object. The default value is **10s**. If you change this default, the **node-status-report-frequency** value is changed to the same value.
- 3 Specify the frequency that the kubelet reports the status of a node associated with this **MachineConfig** object. The default value is **1m**.

The **node-status-update-frequency** parameter works with the **node-monitor-grace-period** parameter.

- The **node-monitor-grace-period** parameter specifies how long OpenShift Container Platform waits after a node associated with a **MachineConfig** object is marked **Unhealthy** if the controller manager does not receive the node heartbeat. Workloads on the node continue to run after this time. If the remote worker node rejoins the cluster after **node-monitor-grace-period** expires, pods continue to run. New pods can be scheduled to that node. The **node-monitor-grace-period** interval is **40s**. The **node-status-update-frequency** value must be lower than the **node-monitor-grace-period** value.



NOTE

Modifying the **node-monitor-grace-period** parameter is not supported.

Tolerations

You can use pod tolerations to mitigate the effects if the on-premise node controller adds a **node.kubernetes.io/unreachable** taint with a **NoExecute** effect to a node it cannot reach.

A taint with the **NoExecute** effect affects pods that are running on the node in the following ways:

- Pods that do not tolerate the taint are queued for eviction.
- Pods that tolerate the taint without specifying a **tolerationSeconds** value in their toleration specification remain bound forever.
- Pods that tolerate the taint with a specified **tolerationSeconds** value remain bound for the specified amount of time. After the time elapses, the pods are queued for eviction.



NOTE

Unless tolerations are explicitly set, Kubernetes automatically adds a toleration for **node.kubernetes.io/not-ready** and **node.kubernetes.io/unreachable** with **tolerationSeconds=300**, meaning that pods remain bound for 5 minutes if either of these taints is detected.

You can delay or avoid pod eviction by configuring pods tolerations with the **NoExecute** effect for the **node.kubernetes.io/unreachable** and **node.kubernetes.io/not-ready** taints.

Example toleration in a pod spec

...
tolerations:

```

- key: "node.kubernetes.io/unreachable"
  operator: "Exists"
  effect: "NoExecute" ❶
- key: "node.kubernetes.io/not-ready"
  operator: "Exists"
  effect: "NoExecute" ❷
  tolerationSeconds: 600 ❸
...

```

- ❶ The **NoExecute** effect without **tolerationSeconds** lets pods remain forever if the control plane cannot reach the node.
- ❷ The **NoExecute** effect with **tolerationSeconds**: 600 lets pods remain for 10 minutes if the control plane marks the node as **Unhealthy**.
- ❸ You can specify your own **tolerationSeconds** value.

Other types of OpenShift Container Platform objects

You can use replica sets, deployments, and replication controllers. The scheduler can reschedule these pods onto other nodes after the node is disconnected for five minutes. Rescheduling onto other nodes can be beneficial for some workloads, such as REST APIs, where an administrator can guarantee a specific number of pods are running and accessible.



NOTE

When working with remote worker nodes, rescheduling pods on different nodes might not be acceptable if remote worker nodes are intended to be reserved for specific functions.

[stateful sets](#) do not get restarted when there is an outage. The pods remain in the **terminating** state until the control plane can acknowledge that the pods are terminated.

To avoid scheduling a pod to a node that does not have access to the same type of persistent storage, OpenShift Container Platform cannot migrate pods that require persistent volumes to other zones in the case of network separation.

Additional resources

- For more information on Daemonsets, see [DaemonSets](#).
- For more information on taints and tolerations, see [Controlling pod placement using node taints](#).
- For more information on configuring **KubeletConfig** objects, see [Creating a KubeletConfig CRD](#).
- For more information on replica sets, see [ReplicaSets](#).
- For more information on deployments, see [Deployments](#).
- For more information on replication controllers, see [Replication controllers](#).
- For more information on the controller manager, see [Kubernetes Controller Manager Operator](#).

CHAPTER 10. WORKER NODES FOR SINGLE-NODE OPENSIFT CLUSTERS

10.1. ADDING WORKER NODES TO SINGLE-NODE OPENSIFT CLUSTERS

Single-node OpenShift clusters reduce the host prerequisites for deployment to a single host. This is useful for deployments in constrained environments or at the network edge. However, sometimes you need to add additional capacity to your cluster, for example, in telecommunications and network edge scenarios. In these scenarios, you can add worker nodes to the single-node cluster.



NOTE

Unlike multi-node clusters, by default all ingress traffic is routed to the single control-plane node, even after adding additional worker nodes.

There are several ways that you can add worker nodes to a single-node cluster. You can add worker nodes to a cluster manually, using [Red Hat OpenShift Cluster Manager](#), or by using the Assisted Installer REST API directly.



IMPORTANT

Adding worker nodes does not expand the cluster control plane, and it does not provide high availability to your cluster. For single-node OpenShift clusters, high availability is handled by failing over to another site. When adding worker nodes to single-node OpenShift clusters, a tested maximum of two worker nodes is recommended. Exceeding the recommended number of worker nodes might result in lower overall performance, including cluster failure.



NOTE

To add worker nodes, you must have access to the OpenShift Cluster Manager. This method is not supported when using the Agent-based installer to install a cluster in a disconnected environment.

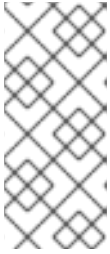
10.1.1. Requirements for installing single-node OpenShift worker nodes

To install a single-node OpenShift worker node, you must address the following requirements:

- **Administration host:** You must have a computer to prepare the ISO and to monitor the installation.
- **Production-grade server:** Installing single-node OpenShift worker nodes requires a server with sufficient resources to run OpenShift Container Platform services and a production workload.

Table 10.1. Minimum resource requirements

Profile	vCPU	Memory	Storage
Minimum	2 vCPU cores	8GB of RAM	100GB

**NOTE**

One vCPU is equivalent to one physical core when simultaneous multithreading (SMT), or hyperthreading, is not enabled. When enabled, use the following formula to calculate the corresponding ratio:

$$(\text{threads per core} \times \text{cores}) \times \text{sockets} = \text{vCPUs}$$

The server must have a Baseboard Management Controller (BMC) when booting with virtual media.

- **Networking:** The worker node server must have access to the internet or access to a local registry if it is not connected to a routable network. The worker node server must have a DHCP reservation or a static IP address and be able to access the single-node OpenShift cluster Kubernetes API, ingress route, and cluster node domain names. You must configure the DNS to resolve the IP address to each of the following fully qualified domain names (FQDN) for the single-node OpenShift cluster:

Table 10.2. Required DNS records

Usage	FQDN	Description
Kubernetes API	api.<cluster_name>.<base_domain>	Add a DNS A/AAAA or CNAME record. This record must be resolvable by clients external to the cluster.
Internal API	api-int.<cluster_name>.<base_domain>	Add a DNS A/AAAA or CNAME record when creating the ISO manually. This record must be resolvable by nodes within the cluster.
Ingress route	*.apps.<cluster_name>.<base_domain>	Add a wildcard DNS A/AAAA or CNAME record that targets the node. This record must be resolvable by clients external to the cluster.

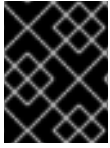
Without persistent IP addresses, communications between the **apiserver** and **etcd** might fail.

Additional resources

- [Minimum resource requirements for cluster installation](#)
- [Recommended practices for scaling the cluster](#)
- [User-provisioned DNS requirements](#)
- [Creating a bootable ISO image on a USB drive](#)
- [Bootting from an ISO image served over HTTP using the Redfish API](#)
- [Deleting nodes from a cluster](#)

10.1.2. Adding worker nodes using the Assisted Installer and OpenShift Cluster Manager

You can add worker nodes to single-node OpenShift clusters that were created on [Red Hat OpenShift Cluster Manager](#) using the [Assisted Installer](#).



IMPORTANT

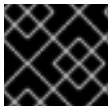
Adding worker nodes to single-node OpenShift clusters is only supported for clusters running OpenShift Container Platform version 4.11 and up.

Prerequisites

- Have access to a single-node OpenShift cluster installed using [Assisted Installer](#).
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Ensure that all the required DNS records exist for the cluster that you are adding the worker node to.

Procedure

1. Log in to [OpenShift Cluster Manager](#) and click the single-node cluster that you want to add a worker node to.
2. Click **Add hosts**, and download the discovery ISO for the new worker node, adding SSH public key and configuring cluster-wide proxy settings as required.
3. Boot the target host using the discovery ISO, and wait for the host to be discovered in the console. After the host is discovered, start the installation.
4. As the installation proceeds, the installation generates pending certificate signing requests (CSRs) for the worker node. When prompted, approve the pending CSRs to complete the installation.
When the worker node is successfully installed, it is listed as a worker node in the cluster web console.



IMPORTANT

New worker nodes will be encrypted using the same method as the original cluster.

Additional resources

- [User-provisioned DNS requirements](#)
- [Approving the certificate signing requests for your machines](#)

10.1.3. Adding worker nodes using the Assisted Installer API

You can add worker nodes to single-node OpenShift clusters using the Assisted Installer REST API. Before you add worker nodes, you must log in to [OpenShift Cluster Manager](#) and authenticate against the API.

10.1.3.1. Authenticating against the Assisted Installer REST API

Before you can use the Assisted Installer REST API, you must authenticate against the API using a JSON web token (JWT) that you generate.

Prerequisites

- Log in to [OpenShift Cluster Manager](#) as a user with cluster creation privileges.
- Install **jq**.

Procedure

1. Log in to [OpenShift Cluster Manager](#) and copy your API token.
2. Set the **\$OFFLINE_TOKEN** variable using the copied API token by running the following command:

```
$ export OFFLINE_TOKEN=<copied_api_token>
```

3. Set the **\$JWT_TOKEN** variable using the previously set **\$OFFLINE_TOKEN** variable:

```
$ export JWT_TOKEN=$(
  curl \
    --silent \
    --header "Accept: application/json" \
    --header "Content-Type: application/x-www-form-urlencoded" \
    --data-urlencode "grant_type=refresh_token" \
    --data-urlencode "client_id=cloud-services" \
    --data-urlencode "refresh_token=${OFFLINE_TOKEN}" \
    "https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token" \
    | jq --raw-output ".access_token"
)
```



NOTE

The JWT token is valid for 15 minutes only.

Verification

- Optional: Check that you can access the API by running the following command:

```
$ curl -s https://api.openshift.com/api/assisted-install/v2/component-versions -H
"Authorization: Bearer ${JWT_TOKEN}" | jq
```

Example output

```
{
  "release_tag": "v2.5.1",
  "versions":
  {
    "assisted-installer": "registry.redhat.io/rhai-tech-preview/assisted-installer-rhel8:v1.0.0-175",
    "assisted-installer-controller": "registry.redhat.io/rhai-tech-preview/assisted-installer-
```

```

reporter-rhel8:v1.0.0-223",
  "assisted-installer-service": "quay.io/app-sre/assisted-service:ac87f93",
  "discovery-agent": "registry.redhat.io/rhai-tech-preview/assisted-installer-agent-
rhel8:v1.0.0-156"
}
}

```

10.1.3.2. Adding worker nodes using the Assisted Installer REST API

You can add worker nodes to clusters using the Assisted Installer REST API.

Prerequisites

- Install the OpenShift Cluster Manager CLI (**ocm**).
- Log in to [OpenShift Cluster Manager](#) as a user with cluster creation privileges.
- Install **jq**.
- Ensure that all the required DNS records exist for the cluster that you are adding the worker node to.

Procedure

1. Authenticate against the Assisted Installer REST API and generate a JSON web token (JWT) for your session. The generated JWT token is valid for 15 minutes only.
2. Set the **\$API_URL** variable by running the following command:

```
$ export API_URL=<api_url> 1
```

- 1 Replace **<api_url>** with the Assisted Installer API URL, for example, <https://api.openshift.com>

3. Import the single-node OpenShift cluster by running the following commands:
 - a. Set the **\$OPENSHIFT_CLUSTER_ID** variable. Log in to the cluster and run the following command:

```
$ export OPENSHIFT_CLUSTER_ID=$(oc get clusterversion -o
jsonpath='{.items[].spec.clusterID}')

```

- b. Set the **\$CLUSTER_REQUEST** variable that is used to import the cluster:

```

$ export CLUSTER_REQUEST=$(jq --null-input --arg openshift_cluster_id
"$OPENSHIFT_CLUSTER_ID" '{
  "api_vip_dnsname": "<api_vip>", 1
  "openshift_cluster_id": $openshift_cluster_id,
  "name": "<openshift_cluster_name>" 2
}')

```


- 1 Replace **<api_vip>** with the hostname for the cluster's API server. This can be the DNS domain for the API server or the IP address of the single node which the worker node can reach. For example, **api.compute-1.example.com**.
- 2 Replace **<openshift_cluster_name>** with the plain text name for the cluster. The cluster name should match the cluster name that was set during the Day 1 cluster installation.

c. Import the cluster and set the **\$CLUSTER_ID** variable. Run the following command:

```
$ CLUSTER_ID=$(curl "$API_URL/api/assisted-install/v2/clusters/import" -H
"Authorization: Bearer ${JWT_TOKEN}" -H 'accept: application/json' -H 'Content-Type:
application/json' \
-d "$CLUSTER_REQUEST" | tee /dev/stderr | jq -r '.id')
```

4. Generate the **InfraEnv** resource for the cluster and set the **\$INFRA_ENV_ID** variable by running the following commands:

- a. Download the pull secret file from Red Hat OpenShift Cluster Manager at console.redhat.com.
- b. Set the **\$INFRA_ENV_REQUEST** variable:

```
export INFRA_ENV_REQUEST=$(jq --null-input \
  --slurpfile pull_secret <path_to_pull_secret_file> \ 1
  --arg ssh_pub_key "$(cat <path_to_ssh_pub_key>)" \ 2
  --arg cluster_id "$CLUSTER_ID" '{
    "name": "<infraenv_name>", 3
    "pull_secret": $pull_secret[0] | tojson,
    "cluster_id": $cluster_id,
    "ssh_authorized_key": $ssh_pub_key,
    "image_type": "<iso_image_type>" 4
  }')
```

- 1 Replace **<path_to_pull_secret_file>** with the path to the local file containing the downloaded pull secret from Red Hat OpenShift Cluster Manager at console.redhat.com.
- 2 Replace **<path_to_ssh_pub_key>** with the path to the public SSH key required to access the host. If you do not set this value, you cannot access the host while in discovery mode.
- 3 Replace **<infraenv_name>** with the plain text name for the **InfraEnv** resource.
- 4 Replace **<iso_image_type>** with the ISO image type, either **full-iso** or **minimal-iso**.

c. Post the **\$INFRA_ENV_REQUEST** to the [/v2/infra-envs](#) API and set the **\$INFRA_ENV_ID** variable:

```
$ INFRA_ENV_ID=$(curl "$API_URL/api/assisted-install/v2/infra-envs" -H "Authorization:
Bearer ${JWT_TOKEN}" -H 'accept: application/json' -H 'Content-Type: application/json'
-d "$INFRA_ENV_REQUEST" | tee /dev/stderr | jq -r '.id')
```

- Get the URL of the discovery ISO for the cluster worker node by running the following command:

```
$ curl -s "$API_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID" -H "Authorization: Bearer ${JWT_TOKEN}" | jq -r '.download_url'
```

Example output

```
https://api.openshift.com/api/assisted-images/images/41b91e72-c33e-42ee-b80f-b5c5bbf6431a?arch=x86_64&image_token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOiJlE2NTYwMjYzNzEsInN1Yil6IjQxYjYkxZTcyLWMzM2UtNDJlZS1iODBmLWI1YzViYmY2NDMxYSJ9.1EX_VGaMNejMhrAvVRBS7PDPIQtOOc8LtG8OukE1a4&type=minimal-iso&version=$VERSION
```

- Download the ISO:

```
$ curl -L -s '<iso_url>' --output rhcos-live-minimal.iso ❶
```

- Replace **<iso_url>** with the URL for the ISO from the previous step.

- Boot the new worker host from the downloaded **rhcos-live-minimal.iso**.
- Get the list of hosts in the cluster that are *not* installed. Keep running the following command until the new host shows up:

```
$ curl -s "$API_URL/api/assisted-install/v2/clusters/$CLUSTER_ID" -H "Authorization: Bearer ${JWT_TOKEN}" | jq -r '.hosts[] | select(.status != "installed").id'
```

Example output

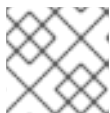
```
2294ba03-c264-4f11-ac08-2f1bb2f8c296
```

- Set the **\$HOST_ID** variable for the new worker node, for example:

```
$ HOST_ID=<host_id> ❶
```

- Replace **<host_id>** with the host ID from the previous step.

- Check that the host is ready to install by running the following command:



NOTE

Ensure that you copy the entire command including the complete **jq** expression.

```
$ curl -s $API_URL/api/assisted-install/v2/clusters/$CLUSTER_ID -H "Authorization: Bearer ${JWT_TOKEN}" | jq 'def host_name($host): if (.suggested_hostname // "") == "" then if (.inventory // "") == "" then "Unknown hostname, please wait"
```

```

        else
            .inventory | fromjson | .hostname
        end
    else
        .suggested_hostname
    end;

def is_notable($validation):
    ["failure", "pending", "error"] | any(. == $validation.status);

def notable_validations($validations_info):
    [
        $validations_info // "{}"
        | fromjson
        | to_entries[].value[]
        | select(is_notable(.))
    ];

{
    "Hosts validations": {
        "Hosts": [
            .hosts[]
            | select(.status != "installed")
            | {
                "id": .id,
                "name": host_name(.),
                "status": .status,
                "notable_validations": notable_validations(.validations_info)
            }
        ]
    },
    "Cluster validations info": {
        "notable_validations": notable_validations(.validations_info)
    }
}
'-r

```

Example output

```

{
  "Hosts validations": {
    "Hosts": [
      {
        "id": "97ec378c-3568-460c-bc22-df54534ff08f",
        "name": "localhost.localdomain",
        "status": "insufficient",
        "notable_validations": [
          {
            "id": "ntp-synced",
            "status": "failure",
            "message": "Host couldn't synchronize with any NTP server"
          },
          {
            "id": "api-domain-name-resolved-correctly",
            "status": "error",
            "message": "Parse error for domain name resolutions result"
          }
        ]
      }
    ]
  }
}

```

```

    },
    {
      "id": "api-int-domain-name-resolved-correctly",
      "status": "error",
      "message": "Parse error for domain name resolutions result"
    },
    {
      "id": "apps-domain-name-resolved-correctly",
      "status": "error",
      "message": "Parse error for domain name resolutions result"
    }
  ]
}
],
},
"Cluster validations info": {
  "notable_validations": []
}
}
}

```

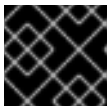
11. When the previous command shows that the host is ready, start the installation using the [/v2/infra-envs/{infra_env_id}/hosts/{host_id}/actions/install](#) API by running the following command:

```

$ curl -X POST -s "$API_URL/api/assisted-install/v2/infra-
envs/$INFRA_ENV_ID/hosts/$HOST_ID/actions/install" -H "Authorization: Bearer
${JWT_TOKEN}"

```

12. As the installation proceeds, the installation generates pending certificate signing requests (CSRs) for the worker node.



IMPORTANT

You must approve the CSRs to complete the installation.

Keep running the following API call to monitor the cluster installation:

```

$ curl -s "$API_URL/api/assisted-install/v2/clusters/$CLUSTER_ID" -H "Authorization: Bearer
${JWT_TOKEN}" | jq '{
  "Cluster day-2 hosts":
    [
      .hosts[]
      | select(.status != "installed")
      | {id, requested_hostname, status, status_info, progress, status_updated_at,
updated_at, infra_env_id, cluster_id, created_at}
    ]
}'

```

Example output

```

{
  "Cluster day-2 hosts": [
    {
      "id": "a1c52dde-3432-4f59-b2ae-0a530c851480",

```

```

    "requested_hostname": "control-plane-1",
    "status": "added-to-existing-cluster",
    "status_info": "Host has rebooted and no further updates will be posted. Please check
console for progress and to possibly approve pending CSRs",
    "progress": {
      "current_stage": "Done",
      "installation_percentage": 100,
      "stage_started_at": "2022-07-08T10:56:20.476Z",
      "stage_updated_at": "2022-07-08T10:56:20.476Z"
    },
    "status_updated_at": "2022-07-08T10:56:20.476Z",
    "updated_at": "2022-07-08T10:57:15.306369Z",
    "infra_env_id": "b74ec0c3-d5b5-4717-a866-5b6854791bd3",
    "cluster_id": "8f721322-419d-4eed-aa5b-61b50ea586ae",
    "created_at": "2022-07-06T22:54:57.161614Z"
  }
}
]
}

```

13. Optional: Run the following command to see all the events for the cluster:

```

$ curl -s "$API_URL/api/assisted-install/v2/events?cluster_id=$CLUSTER_ID" -H
"Authorization: Bearer ${JWT_TOKEN}" | jq -c '.[] | {severity, message, event_time, host_id}'

```

Example output

```

{"severity":"info","message":"Host compute-0: updated status from insufficient to known (Host
is ready to be installed)","event_time":"2022-07-08T11:21:46.346Z","host_id":"9d7b3b44-
1125-4ad0-9b14-76550087b445"}
{"severity":"info","message":"Host compute-0: updated status from known to installing
(Installation is in progress)","event_time":"2022-07-08T11:28:28.647Z","host_id":"9d7b3b44-
1125-4ad0-9b14-76550087b445"}
{"severity":"info","message":"Host compute-0: updated status from installing to installing-in-
progress (Starting installation)","event_time":"2022-07-
08T11:28:52.068Z","host_id":"9d7b3b44-1125-4ad0-9b14-76550087b445"}
{"severity":"info","message":"Uploaded logs for host compute-0 cluster 8f721322-419d-4eed-
aa5b-61b50ea586ae","event_time":"2022-07-08T11:29:47.802Z","host_id":"9d7b3b44-1125-
4ad0-9b14-76550087b445"}
{"severity":"info","message":"Host compute-0: updated status from installing-in-progress to
added-to-existing-cluster (Host has rebooted and no further updates will be posted. Please
check console for progress and to possibly approve pending CSRs)","event_time":"2022-07-
08T11:29:48.259Z","host_id":"9d7b3b44-1125-4ad0-9b14-76550087b445"}
{"severity":"info","message":"Host: compute-0, reached installation stage
Rebooting","event_time":"2022-07-08T11:29:48.261Z","host_id":"9d7b3b44-1125-4ad0-9b14-
76550087b445"}

```

14. Log in to the cluster and approve the pending CSRs to complete the installation.

Verification

- Check that the new worker node was successfully added to the cluster with a status of **Ready**:

```

$ oc get nodes

```

Example output

NAME	STATUS	ROLES	AGE	VERSION
control-plane-1.example.com	Ready	master,worker	56m	v1.33.4
compute-1.example.com	Ready	worker	11m	v1.33.4

Additional resources

- [User-provisioned DNS requirements](#)
- [Approving the certificate signing requests for your machines](#)

10.1.4. Adding worker nodes to single-node OpenShift clusters manually

You can add a worker node to a single-node OpenShift cluster manually by booting the worker node from Red Hat Enterprise Linux CoreOS (RHCOS) ISO and by using the cluster **worker.ign** file to join the new worker node to the cluster.

Prerequisites

- Install a single-node OpenShift cluster on bare metal.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Ensure that all the required DNS records exist for the cluster that you are adding the worker node to.

Procedure

1. Set the OpenShift Container Platform version:

```
$ OCP_VERSION=<ocp_version> 1
```

- 1 Replace **<ocp_version>** with the current version, for example, **latest-4.20**

2. Set the host architecture:

```
$ ARCH=<architecture> 1
```

- 1 Replace **<architecture>** with the target host architecture, for example, **aarch64** or **x86_64**.

3. Get the **worker.ign** data from the running single-node cluster by running the following command:

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

4. Host the **worker.ign** file on a web server accessible from your network.

5. Download the OpenShift Container Platform installer and make it available for use by running the following commands:

```
$ curl -k https://mirror.openshift.com/pub/openshift-
v4/clients/ocp/$OCP_VERSION/openshift-install-linux.tar.gz > openshift-install-linux.tar.gz
```

```
$ tar zxvf openshift-install-linux.tar.gz
```

```
$ chmod +x openshift-install
```

6. Retrieve the RHCOS ISO URL:

```
$ ISO_URL=$(./openshift-install coreos print-stream-json | grep location | grep $ARCH | grep
iso | cut -d\" -f4)
```

7. Download the RHCOS ISO:

```
$ curl -L $ISO_URL -o rhcos-live.iso
```

8. Use the RHCOS ISO and the hosted **worker.ign** file to install the worker node:

- a. Boot the target host with the RHCOS ISO and your preferred method of installation.
- b. When the target host has booted from the RHCOS ISO, open a console on the target host.
- c. If your local network does not have DHCP enabled, you need to create an ignition file with the new hostname and configure the worker node static IP address before running the RHCOS installation. Perform the following steps:
 - i. Configure the worker host network connection with a static IP. Run the following command on the target host console:

```
$ nmcli con mod <network_interface> ipv4.method manual /
ipv4.addresses <static_ip> ipv4.gateway <network_gateway> ipv4.dns <dns_server>
/
802-3-ethernet.mtu 9000
```

where:

<static_ip>

Is the host static IP address and CIDR, for example, **10.1.101.50/24**

<network_gateway>

Is the network gateway, for example, **10.1.101.1**

- ii. Activate the modified network interface:

```
$ nmcli con up <network_interface>
```

- iii. Create a new ignition file **new-worker.ign** that includes a reference to the original **worker.ign** and an additional instruction that the **coreos-installer** program uses to populate the **/etc/hostname** file on the new worker host. For example:

```

{
  "ignition":{
    "version":"3.2.0",
    "config":{
      "merge":[
        {
          "source":"<hosted_worker_ign_file>" ❶
        }
      ]
    }
  },
  "storage":{
    "files":[
      {
        "path":"/etc/hostname",
        "contents":{
          "source":"data:,<new_fqdn>" ❷
        },
        "mode":420,
        "overwrite":true,
        "path":"/etc/hostname"
      }
    ]
  }
}

```

❶ **<hosted_worker_ign_file>** is the locally accessible URL for the original **worker.ign** file. For example, <http://webserver.example.com/worker.ign>

❷ **<new_fqdn>** is the new FQDN that you set for the worker node. For example, **new-worker.example.com**.

- iv. Host the **new-worker.ign** file on a web server accessible from your network.
- v. Run the following **coreos-installer** command, passing in the **ignition-url** and hard disk details:

```
$ sudo coreos-installer install --copy-network /
--ignition-url=<new_worker_ign_file> <hard_disk> --insecure-ignition
```

where:

<new_worker_ign_file>

is the locally accessible URL for the hosted **new-worker.ign** file, for example, <http://webserver.example.com/new-worker.ign>

<hard_disk>

is the hard disk where you install RHCOS, for example, **/dev/sda**

- d. For networks that have DHCP enabled, you do not need to set a static IP. Run the following **coreos-installer** command from the target host console to install the system:

```
$ coreos-installer install --ignition-url=<hosted_worker_ign_file> <hard_disk>
```


- e. To manually enable DHCP, apply the following **NMStateConfig** CR to the single-node OpenShift cluster:

```
apiVersion: agent-install.openshift.io/v1
kind: NMStateConfig
metadata:
  name: nmstateconfig-dhcp
  namespace: example-sno
  labels:
    nmstate_config_cluster_name: <nmstate_config_cluster_label>
spec:
  config:
    interfaces:
      - name: eth0
        type: ethernet
        state: up
        ipv4:
          enabled: true
          dhcp: true
        ipv6:
          enabled: false
    interfaces:
      - name: "eth0"
        macAddress: "AA:BB:CC:DD:EE:11"
```



IMPORTANT

The **NMStateConfig** CR is required for successful deployments of worker nodes with static IP addresses and for adding a worker node with a dynamic IP address if the single-node OpenShift was deployed with a static IP address. The cluster network DHCP does not automatically set these network settings for the new worker node.

9. As the installation proceeds, the installation generates pending certificate signing requests (CSRs) for the worker node. When prompted, approve the pending CSRs to complete the installation.
10. When the install is complete, reboot the host. The host joins the cluster as a new worker node.

Verification

- Check that the new worker node was successfully added to the cluster with a status of **Ready**:

```
$ oc get nodes
```

Example output

NAME	STATUS	ROLES	AGE	VERSION
control-plane-1.example.com	Ready	master,worker	56m	v1.33.4
compute-1.example.com	Ready	worker	11m	v1.33.4

Additional resources

- [User-provisioned DNS requirements](#)
- [Approving the certificate signing requests for your machines](#)

10.1.5. 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.

- 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](#)

CHAPTER 11. NODE METRICS DASHBOARD

The node metrics dashboard is a visual analytics dashboard that helps you identify potential pod scaling issues.

11.1. ABOUT THE NODE METRICS DASHBOARD

The node metrics dashboard enables administrative and support team members to monitor metrics related to pod scaling, including scaling limits used to diagnose and troubleshoot scaling issues. Particularly, you can use the visual analytics displayed through the dashboard to monitor workload distributions across nodes. Insights gained from these analytics help you determine the health of your CRI-O and Kubelet system components as well as identify potential sources of excessive or imbalanced resource consumption and system instability.

The dashboard displays visual analytics widgets organized into the following categories:

Critical

Includes visualizations that can help you identify node issues that could result in system instability and inefficiency

Outliers

Includes histograms that visualize processes with runtime durations that fall outside of the 95th percentile

Average durations

Helps you track change in the time that system components take to process operations

Number of operations

Displays visualizations that help you identify changes in the number of operations being run, which in turn helps you determine the load balance and efficiency of your system

11.2. ACCESSING THE NODE METRICS DASHBOARD

You can access the node metrics dashboard from the **Administrator** perspective.

Procedure

1. Expand the **Observe** menu option and select **Dashboards**.
2. Under the **Dashboard** filter, select **Node cluster**.



NOTE

If no data appears in the visualizations under the **Critical** category, no critical anomalies were detected. The dashboard is working as intended.

11.3. IDENTIFY METRICS FOR INDICATING OPTIMAL NODE RESOURCE USAGE

The node metrics dashboard is organized into four categories: **Critical**, **Outliers**, **Average durations**, and **Number of Operations**. The metrics in the **Critical** category help you indicate optimal node resource usage. These metrics include:

- Top 3 containers with the most OOM kills in the last day

- Failure rate for image pulls in the last hour
- Nodes with system reserved memory utilization > 80%
- Nodes with Kubelet system reserved memory utilization > 50%
- Nodes with CRI-O system reserved memory utilization > 50%
- Nodes with system reserved CPU utilization > 80%
- Nodes with Kubelet system reserved CPU utilization > 50%
- Nodes with CRI-O system reserved CPU utilization > 50%

11.3.1. Top 3 containers with the most OOM kills in the last day

The **Top 3 containers with the most OOM kills in the last day** query fetches details regarding the top three containers that have experienced the most Out-Of-Memory (OOM) kills in the previous day.

Example default query

```
topk(3, sum(increase(container_runtime_crio_containers_oom_count_total[1d])) by (name))
```

OOM kills force the system to terminate some processes due to low memory. Frequent OOM kills can hinder the functionality of the node and even the entire Kubernetes ecosystem. Containers experiencing frequent OOM kills might be consuming more memory than they should, which causes system instability.

Use this metric to identify containers that are experiencing frequent OOM kills and investigate why these containers are consuming an excessive amount of memory. Adjust the resource allocation if necessary and consider resizing the containers based on their memory usage. You can also review the metrics under the **Outliers**, **Average durations**, and **Number of operations** categories to gain further insights into the health and stability of your nodes.

11.3.2. Failure rate for image pulls in the last hour

The **Failure rate for image pulls in the last hour** query divides the total number of failed image pulls by the sum of successful and failed image pulls to provide a ratio of failures.

Example default query

```
rate(container_runtime_crio_image_pulls_failure_total[1h]) /  
(rate(container_runtime_crio_image_pulls_success_total[1h]) +  
rate(container_runtime_crio_image_pulls_failure_total[1h]))
```

Understanding the failure rate of image pulls is crucial for maintaining the health of the node. A high failure rate might indicate networking issues, storage problems, misconfigurations, or other issues that could disrupt pod density and the deployment of new containers.

If the outcome of this query is high, investigate possible causes such as network connections, the availability of remote repositories, node storage, and the accuracy of image references. You can also review the metrics under the **Outliers**, **Average durations**, and **Number of operations** categories to gain further insights.

11.3.3. Nodes with system reserved memory utilization > 80%

The **Nodes with system reserved memory utilization > 80%** query calculates the percentage of system reserved memory that is utilized for each node. The calculation divides the total resident set size (RSS) by the total memory capacity of the node subtracted from the allocatable memory. RSS is the portion of the system's memory occupied by a process that is held in main memory (RAM). Nodes are flagged if their resulting value equals or exceeds an 80% threshold.

Example default query

```
sum by (node) (container_memory_rss{id="/system.slice"}) / sum by (node)
(kube_node_status_capacity{resource="memory"} -
kube_node_status_allocatable{resource="memory"}) * 100 >= 80
```

System reserved memory is crucial for a Kubernetes node as it is utilized to run system daemons and Kubernetes system daemons. System reserved memory utilization that exceeds 80% indicates that the system and Kubernetes daemons are consuming too much memory and can suggest node instability that could affect the performance of running pods. Excessive memory consumption can cause Out-of-Memory (OOM) killers that can terminate critical system processes to free up memory.

If a node is flagged by this metric, identify which system or Kubernetes processes are consuming excessive memory and take appropriate actions to mitigate the situation. These actions may include scaling back non-critical processes, optimizing program configurations to reduce memory usage, or upgrading node systems to hardware with greater memory capacity. You can also review the metrics under the **Outliers**, **Average durations**, and **Number of operations** categories to gain further insights into node performance.

11.3.4. Nodes with Kubelet system reserved memory utilization > 50%

The **Nodes with Kubelet system reserved memory utilization > 50%** query indicates nodes where the Kubelet's system reserved memory utilization exceeds 50%. The query examines the memory that the Kubelet process itself is consuming on a node.

Example default query

```
sum by (node) (container_memory_rss{id="/system.slice/kubelet.service"}) / sum by (node)
(kube_node_status_capacity{resource="memory"} -
kube_node_status_allocatable{resource="memory"}) * 100 >= 50
```

This query helps you identify any possible memory pressure situations in your nodes that could affect the stability and efficiency of node operations. Kubelet memory utilization that consistently exceeds 50% of the system reserved memory, indicate that the system reserved settings are not configured properly and that there is a high risk of the node becoming unstable.

If this metric is highlighted, review your configuration policy and consider adjusting the system reserved settings or the resource limits settings for the Kubelet. Additionally, if your Kubelet memory utilization consistently exceeds half of your total reserved system memory, examine metrics under the **Outliers**, **Average durations**, and **Number of operations** categories to gain further insights for more precise diagnostics.

11.3.5. Nodes with CRI-O system reserved memory utilization > 50%

The **Nodes with CRI-O system reserved memory utilization > 50%** query calculates all nodes where the percentage of used memory reserved for the CRI-O system is greater than or equal to 50%. In this

case, memory usage is defined by the resident set size (RSS), which is the portion of the CRI-O system's memory held in RAM.

Example default query

```
sum by (node) (container_memory_rss{id="/system.slice/crio.service"}) / sum by (node)
(kube_node_status_capacity{resource="memory"} -
kube_node_status_allocatable{resource="memory"}) * 100 >= 50
```

This query helps you monitor the status of memory reserved for the CRI-O system on each node. High utilization could indicate a lack of available resources and potential performance issues. If the memory reserved for the CRI-O system exceeds the advised limit of 50%, it indicates that half of the system reserved memory is being used by CRI-O on a node.

Check memory allocation and usage and assess whether memory resources need to be shifted or increased to prevent possible node instability. You can also examine the metrics under the **Outliers**, **Average durations**, and **Number of operations** categories to gain further insights.

11.3.6. Nodes with System Reserved CPU Utilization > 80%

The **Nodes with system reserved CPU utilization > 80%** query identifies nodes where the system-reserved CPU utilization is more than 80%. The query focuses on the system-reserved capacity to calculate the rate of CPU usage in the last 5 minutes and compares that to the CPU resources available on the nodes. If the ratio exceeds 80%, the node's result is displayed in the metric.

Example default query

```
sum by (node) (rate(container_cpu_usage_seconds_total{id="/system.slice"}[5m]) * 100) / sum by
(node) (kube_node_status_capacity{resource="cpu"} -
kube_node_status_allocatable{resource="cpu"}) >= 80
```

This query indicates a critical level of system-reserved CPU usage, which can lead to resource exhaustion. High system-reserved CPU usage can result in the inability of the system processes (including the Kubelet and CRI-O) to adequately manage resources on the node. This query can indicate excessive system processes or misconfigured CPU allocation.

Potential corrective measures include rebalancing workloads to other nodes or increasing the CPU resources allocated to the nodes. Investigate the cause of the high system CPU utilization and review the corresponding metrics in the **Outliers**, **Average durations**, and **Number of operations** categories for additional insights into the node's behavior.

11.3.7. Nodes with Kubelet system reserved CPU utilization > 50%

The **Nodes with Kubelet system reserved CPU utilization > 50%** query calculates the percentage of the CPU that the Kubelet system is currently using from system reserved.

Example default query

```
sum by (node) (rate(container_cpu_usage_seconds_total{id="/system.slice/kubelet.service"}[5m]) *
100) / sum by (node) (kube_node_status_capacity{resource="cpu"} -
kube_node_status_allocatable{resource="cpu"}) >= 50
```

The Kubelet uses the system reserved CPU for its own operations and for running critical system services. For the node's health, it is important to ensure that system reserve CPU usage does not

exceed the 50% threshold. Exceeding this limit could indicate heavy utilization or load on the Kubelet, which affects node stability and potentially the performance of the entire Kubernetes cluster.

If any node is displayed in this metric, the Kubelet and the system overall are under heavy load. You can reduce overload on a particular node by balancing the load across other nodes in the cluster. Check other query metrics under the **Outliers**, **Average durations**, and **Number of operations** categories to gain further insights and take necessary corrective action.

11.3.8. Nodes with CRI-O system reserved CPU utilization > 50%

The **Nodes with CRI-O system reserved CPU utilization > 50%** query identifies nodes where the CRI-O system reserved CPU utilization has exceeded 50% in the last 5 minutes. The query monitors CPU resource consumption by CRI-O, your container runtime, on a per-node basis.

Example default query

```
sum by (node) (rate(container_cpu_usage_seconds_total{id="/system.slice/crio.service"}[5m]) * 100)
/ sum by (node) (kube_node_status_capacity{resource="cpu"} -
kube_node_status_allocatable{resource="cpu"}) >= 50
```

This query allows for quick identification of abnormal start times that could negatively impact pod performance. If this query returns a high value, your pod start times are slower than usual, which suggests potential issues with the kubelet, pod configuration, or resources.

Investigate further by checking your pod configurations and allocated resources. Make sure that they align with your system capabilities. If you still see high start times, explore metrics panels from other categories on the dashboard to determine the state of your system components.

11.4. CUSTOMIZING DASHBOARD QUERIES

You can customize the default queries used to build the node metrics dashboard.

Procedure

1. Choose a metric and click **Inspect** to navigate into the data. This page displays the metric in detail, including an expanded visualization of the results of the query, the Prometheus query used to analyze the data, and the data subset used in the query.
2. Make any required changes to the query parameters.
3. Optional: Click **Add query** to run additional queries against the data.
4. Click **Run query** to rerun the query using your specified parameters.

CHAPTER 12. MANAGE SECURE SIGNATURES WITH SIGSTORE

You can use [sigstore](#) with OpenShift Container Platform to improve supply chain security.

12.1. ABOUT SIGSTORE

The sigstore project enables developers to sign-off on what they build and administrators to verify signatures and monitor workflows at scale. With the sigstore project, signatures can be stored in the same registry as the build images. A second server is not needed. The identity piece of a signature is tied to the OpenID Connect (OIDC) identity through the Fulcio certificate authority, which simplifies the signature process by allowing key-less signing. Additionally, sigstore includes Rekor, which records signature metadata to an immutable, tamper-resistant ledger.

You can use the **ClusterImagePolicy** and **ImagePolicy** custom resource (CR) objects to enable and configure sigstore support at the cluster or namespace scope. These objects specify the images and repositories to be verified and how the signatures must be verified.

12.2. ABOUT CONFIGURING SIGSTORE SUPPORT

You can use the **ClusterImagePolicy** and **ImagePolicy** custom resource (CR) objects to enable and configure sigstore support for the entire cluster or a specific namespace. These objects contain a policy that specifies the images and repositories to be verified by using sigstore tooling and how the signatures must be verified.

- Cluster image policy. A cluster image policy object enables a cluster administrator to configure a sigstore signature verification policy for the entire cluster. When enabled, the Machine Config Operator (MCO) watches the **ClusterImagePolicy** object and updates the `/etc/containers/policy.json` and `/etc/containers/registries.d/sigstore-registries.yaml` files on all nodes in the cluster.

IMPORTANT

The default **openshift** cluster image policy provides sigstore support for the required OpenShift Container Platform images. You must not remove or modify this cluster image policy object. This cluster image policy is Technology Preview and is active only in clusters that have enabled Technology Preview features. Cluster image policy names beginning with **openshift** are reserved for future system use.

If registry mirrors are configured for the OpenShift Container Platform release image repositories, **quay.io/openshift-release-dev/ocp-release** and **quay.io/openshift-release-dev/ocp-v4.0-art-dev**, before enabling the Technology Preview feature set, you must mirror the sigstore signatures for the OpenShift Container Platform release images into your mirror registry. Otherwise, the default **openshift** cluster image policy, which enforces signature verification for the release repository, blocks the ability of the Cluster Version Operator to move the CVO pod to new nodes, preventing the node update that results from the feature set change.

You can use the **oc image mirror** command to mirror the signatures. For example:

```
$ oc image mirror quay.io/openshift-release-dev/ocp-release:sha256-
1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef.si
g \
mirror.com/image/repo:sha256-
1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef.si
g
```

- Image policy. An image policy enables a cluster administrator or application developer to configure a sigstore signature verification policy for a specific namespace. The MCO watches an **ImagePolicy** instance in different namespaces and creates or updates the **/etc/crio/policies/<namespace>.json** and **/etc/containers/registries.d/sigstore-registries.yaml** files on all nodes in the cluster.

If the image or repository in an image policy is nested under one of the images or repositories in a cluster image policy, only the policy from cluster image policy is applied. For example, if an image policy specifies **example.com/global/image**, and the cluster image policy specifies **example.com/global**, the namespace uses the policy from the cluster image policy. The image policy object is created and shows an error similar to the following message:

Example image policy with a conflicting image identity

```
API Version: config.openshift.io/v1
Kind:      ImagePolicy
Name:      p0
Namespace: mynamespace
# ...
Status:
  Conditions:
    Message: has conflicting scope(s) ["example.com/global/image"] that equal to or nest
inside existing clusterimagepolicy, only policy from clusterimagepolicy scope(s) will be applied
    Reason: ConflictScopes
# ...
```

12.2.1. About cluster and image policy parameters

The following parameters apply to cluster and image policies. For information on using these parameters, see "Creating a cluster image policy CR" and "Creating an image policy CR."

scopes

Defines a list of repositories and images assigned to a policy. You must list at least one of the following scopes:

- An individual image, by using a tag or digest, such as **example.com/namespace/image:latest**
- A repository, by omitting the tag or digest, such as **example.com**
- A repository namespace, such as **example.com/namespace/**
- A registry host, by specifying only the host name and port number or a wildcard expression starting with *****, such as ***.example.com**

If multiple scopes match a single scope in the same a cluster or image policy, the policy for only the most specific scope applies.

If a scoped image or repository in an image policy is nested under one of the scoped images or repositories in a cluster image policy, only the policy from cluster image policy is applied. However, the image policy object is created. For example, if an image policy specifies **example.com/global/image**, and the cluster image policy specifies **example.com/global**, the namespace inherits the policy from the cluster image policy.

policy

Contains configuration to allow images from the sources listed in **scopes** to be verified, and defines how images not matching the verification policy are treated. You must configure a **rootOfTrust** and optionally, a **signedIdentity**.

- **rootOfTrust**: Specifies the root of trust for the policy. Configure either a public key, a Bring Your Own Public Key Infrastructure (BYOPKI) certificate, or a [Fulcio certificate](#).
 - **publicKey**: Indicates that the policy relies on a sigstore public key. You must specify a base64-encoded PEM format public key. You can optionally include [Rekor verification](#).
 - **PKI** Indicates that the policy relies on a certificate from your own public key infrastructure (PKI) that is compatible with Cosign Bring Your Own Public Key Infrastructure (BYOPKI) verification. You must specify a base64-encoded PEM format public key. BYOPKI enables you to validate container images using an existing X.509 certificate while aligning with Cosign's bring-your-own PKI signing workflow.



IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

- **FulcioCAWithRekor:** Indicates that the policy is based on a Fulcio certificate. You must specify the following parameters:
 - A base64-encoded PEM-format Fulcio CA
 - An OpenID Connect (OIDC) issuer
 - The email of the Fulcio authentication configuration
 - The [Rekor verification](#)
- **signedIdentity:** Specifies the approach used to verify the image in the signature and the actual image itself. To configure a signed identity, you must specify one of the following parameters as the match policy:
 - **MatchRepoDigestOrExact.** The image referenced in the signature must be in the same repository as the image itself. If the image carries a tag, the image referenced in the signature must match exactly. This is the default.
 - **MatchRepository.** The image referenced in the signature must be in the same repository as the image itself. If the image carries a tag, the image referenced in the signature does not need to match exactly. This is useful to pull an image that contains the **latest** tag if the image is signed with a tag specifying an exact image version.
 - **ExactRepository.** The image referenced in the signature must be in the same repository that is specified by the **exactRepository** parameter. The **exactRepository** parameter must be specified.
 - **RemapIdentity.** If the scoped repository or image matches a specified **prefix**, that prefix is replaced by a specified **signedPrefix**. If the image identity does not match, the **prefix** is unchanged and no remapping takes place. This option can be used when verifying signatures for a mirror of some other repository namespace that preserves the vendor's repository structure.
 The **prefix** and **signedPrefix** can be either **host[:port]** values that match the exact **host[:port]** string, repository namespaces, or repositories. The **prefix** and **signedPrefix** must not contain tags or digests. For example, to specify a single repository, use **example.com/library/busybox** and not **busybox**. To specify the parent namespace of **example.com/library/busybox**, you can use **example.com/library**.

You must specify the following parameters:

- **prefix:** Specifies the image prefix to be matched.
- **signedPrefix:** Specifies the image prefix to be remapped, if needed.

12.2.2. About modifying or removing image policies

You can modify or remove a cluster image policy or an image policy by using the same commands as any other custom resource (CR) object.

You can modify an existing policy by editing the policy YAML and running an **oc apply** command on the file or directly editing the **ClusterImagePolicy** or **ImagePolicy** object. Both methods apply the changes in the same manner.

You can create multiple policies for a cluster or namespace. This allows you to create different policies for different images or repositories.

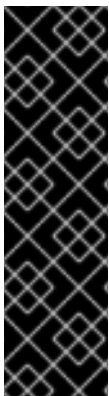
You can remove a policy by deleting the **ClusterImagePolicy** and **ImagePolicy** objects.

12.3. CREATING A CLUSTER IMAGE POLICY CR

A **ClusterImagePolicy** custom resource (CR) enables a cluster administrator to configure a sigstore signature verification policy for the entire cluster. When enabled, the Machine Config Operator (MCO) watches the **ClusterImagePolicy** object and updates the `/etc/containers/policy.json` and `/etc/containers/registries.d/sigstore-registries.yaml` files on all the nodes in the cluster.

The following example shows general guidelines on how to configure a **ClusterImagePolicy** object. For more details on the parameters, see "About cluster and image policy parameters."

The default **openshift** cluster image policy provides sigstore support for the required OpenShift Container Platform images. This cluster image policy is active only in clusters that have enabled Technology Preview features. You must not remove or modify this cluster image policy object. Cluster image policy names beginning with **openshift** are reserved for future system use.



IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

Prerequisites

- You have a sigstore-supported public key infrastructure (PKI) key, a Bring Your Own Public Key Infrastructure (BYOPKI) certificate, or provide a [Cosign public and private key pair](#) for signing operations.
- You have a signing process in place to sign your images.
- You have access to a registry that supports Cosign signatures, if you are using Cosign signatures.
- If registry mirrors are configured for the OpenShift Container Platform release image repositories, **quay.io/openshift-release-dev/ocp-release** and **quay.io/openshift-release-dev/ocp-v4.0-art-dev**, before enabling the Technology Preview feature set, you must mirror the

sigstore signatures for the OpenShift Container Platform release images into your mirror registry. Otherwise, the default **openshift** cluster image policy, which enforces signature verification for the release repository, blocks the ability of the Cluster Version Operator to move the CVO pod to new nodes, preventing the node update that results from the feature set change.

You can use the **oc image mirror** command to mirror the signatures. For example:

```
$ oc image mirror quay.io/openshift-release-dev/ocp-release:sha256-
1234567890abcdef1234567890abcdef1234567890abcdef.sig \
mirror.com/image/repo:sha256-
1234567890abcdef1234567890abcdef1234567890abcdef.sig
```

- If you are using a BYOPKI certificate as the root of trust, you enabled the required Technology Preview features for your cluster by editing the **FeatureGate** CR named **cluster**:

```
$ oc edit featuregate cluster
```

Example FeatureGate CR

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

- 1** Enables the required **SigstoreImageVerificationPKI** feature.



WARNING

Enabling the **TechPreviewNoUpgrade** feature set on your cluster cannot be undone and prevents minor version updates. This feature set allows you to enable these Technology Preview features on test clusters, where you can fully test them. Do not enable this feature set on production clusters.

Procedure

1. Create a cluster image policy object similar to the following examples. See "About image policy parameters" for specific details on these parameters.

Example cluster image policy object with a public key policy and the **MatchRepoDigestOrExact** match policy

```
apiVersion: config.openshift.io/v1
kind: ClusterImagePolicy 1
metadata:
  name: p1
```

```
spec:
  scopes: ❷
    - example.com
  policy: ❸
    rootOfTrust: ❹
      policyType: PublicKey ❺
      publicKey:
        keyData: a2V5RGF0YQ== ❻
        rekorKeyData: cmVrb3JLZXIEYXRh ❼
      signedIdentity: ❽
      matchPolicy: MatchRepoDigestOrExact
```

- ❶ Creates a **ClusterImagePolicy** object.
- ❷ Defines a list of repositories or images assigned to this policy. In a cluster image policy, make sure that the policy does not block the deployment of the OpenShift Container Platform images in the **quay.io/openshift-release-dev/ocp-release** and **quay.io/openshift-release-dev/ocp-v4.0-art-dev** repositories. Images in these repositories are required for cluster operation.
- ❸ Specifies the parameters that define how the images are verified.
- ❹ Defines a root of trust for the policy.
- ❺ Specifies the policy types that define the root of trust, either a public key, a BYOPKI certificate, or a [Fulcio certificate](#). This example uses a public key with Rekor verification.
- ❻ For a public key policy, specifies a base64-encoded public key in the PEM format. The maximum length is 8192 characters.
- ❼ Optional: Specifies a base64-encoded Rekor public key in the PEM format. The maximum length is 8192 characters.
- ❽ Optional: Specifies one of the following processes to verify the identity in the signature and the actual image identity:
 - **MatchRepoDigestOrExact**.
 - **MatchRepository**.
 - **ExactRepository**. The **exactRepository** parameter must be specified.
 - **RemapIdentity**. The **prefix** and **signedPrefix** parameters must be specified.

Example cluster image policy object for a BYOPKI policy and the **MatchRepository** match policy

```
apiVersion: config.openshift.io/v1alpha1
kind: ClusterImagePolicy ❶
metadata:
  name: pki-policy
spec:
  scopes:
    - example.io ❷
```



```

policy: 3
  rootOfTrust: 4
    policyType: PKI 5
    pki: 6
      caRootsData: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk....URS0tLS0t
      caIntermediatesData:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1J....IDQVRFLS0tLS0=
      pkiCertificateSubject: 7
        email: email@example.com
        hostname: myhost.example.com
    signedIdentity:
      matchPolicy: MatchRepository 8

```

- 1 Creates a **ClusterImagePolicy** object.
- 2 Defines a list of repositories or images assigned to this policy. In a cluster image policy, make sure that the policy does not block the deployment of the OpenShift Container Platform images in the **quay.io/openshift-release-dev/ocp-release** and **quay.io/openshift-release-dev/ocp-v4.0-art-dev** repositories. Images in these repositories are required for cluster operation.
- 3 Specifies the parameters that define how the images are verified.
- 4 Defines a root of trust for the policy.
- 5 Specifies the policy types that define the root of trust, either a public key, a BYOPKI certificate, or a [Fulcio certificate](#). This example uses a BYOPKI certificate.
- 6 For a BYOPKI certificate, specify **caRootsData**. This parameter specifies a base64-encoded CA root certificate in the PEM format. The maximum length is 8192 characters. Optionally with **caIntermediatesData**, specifies a base64-encoded intermediate CA root certificate in the PEM format. The maximum length is 8192 characters.
- 7 Specifies a subject alternative name (SAN) to authenticate the user's identity by using a hostname and an email address:
 - **email**. Specifies the email address specified when the certificate was generated.
 - **hostname**. Specifies the hostname specified when the certificate was generated.
- 8 For a BYOPKI certificate, specify the **MatchRepository** parameter to verify the identity in the signature and the actual image identity. The default signed identity is **matchRepoDigestOrExact**, which requires a digest reference in the signature identity for verification. The signature identity in this case uses a repository reference, and does not include the image digest.

Example cluster image policy object with a Fulcio certificate policy and the remapIdentity match policy

```

apiVersion: config.openshift.io/v1
kind: ClusterImagePolicy 1
metadata:
  name: p1
spec:

```

```

scopes: 2
  - example.com
policy: 3
  rootOfTrust: 4
    policyType: FulcioCAWithRekor 5
    fulcioCAWithRekor: 6
      fulcioCAData: a2V5RGF0YQ==
      fulcioSubject:
        oidcIssuer: "https://expected.OIDC.issuer/"
        signedEmail: "expected-signing-user@example.com"
      rekorKeyData: cmVrb3JLZXIEYXRh 7
  signedIdentity:
    matchPolicy: RemapIdentity 8
    remapIdentity:
      prefix: example.com 9
      signedPrefix: mirror-example.com 10

```

- 1 Creates a **ClusterImagePolicy** object.
- 2 Defines a list of repositories or images assigned to this policy. In a cluster image policy, make sure that the policy does not block the deployment of the OpenShift Container Platform images in the **quay.io/openshift-release-dev/ocp-release** and **quay.io/openshift-release-dev/ocp-v4.0-art-dev** repositories. Images in these repositories are required for cluster operation.
- 3 Specifies the parameters that define how the images are verified.
- 4 Defines a root of trust for the policy.
- 5 Specifies the policy types that define the root of trust, either a public key, a BYOPKI certificate, or a [Fulcio certificate](#). This example uses a Fulcio certificate with required Rekor verification.
- 6 For a Fulcio certificate policy, the following parameters are required:
 - **fulcioCAData**: Specifies a base64-encoded Fulcio certificate in the PEM format. The maximum length is 8192 characters.
 - **fulcioSubject**: Specifies the OIDC issuer and the email of the Fulcio authentication configuration.
- 7 Specifies a base64-encoded Rekor public key in the PEM format. This parameter is required when the **policyType** is **FulcioCAWithRekor**. The maximum length is 8192 characters.
- 8 Optional: Specifies one of the following processes to verify the identity in the signature and the actual image identity.
 - **MatchRepoDigestOrExact**.
 - **MatchRepository**.
 - **ExactRepository**. The **exactRepository** parameter must be specified.
 - **RemapIdentity**. The **prefix** and **signedPrefix** parameters must be specified.

- 9 For the **remapIdentity** match policy, specifies the prefix that should be matched against the scoped image prefix. If the two match, the scoped image prefix is replaced with the
- 10 For the **remapIdentity** match policy, specifies the image prefix to be remapped, if needed. The maximum length is 512 characters.

2. Create the cluster image policy object:

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

The Machine Config Operator (MCO) updates the machine config pools (MCP) in your cluster. Scheduling on each node is disabled as the change is being applied.

Verification

- After the nodes in your cluster are updated, you can verify that the cluster image policy has been configured:

a. Start a debug pod for the node by running the following command:

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

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

```
sh-5.1# chroot /host/
```

c. Examine the **policy.json** file by running the following command:

```
sh-5.1# cat /etc/containers/policy.json
```

Example output for the cluster image policy object with a public key showing the new cluster image policy

```
# ...
  "transports": {
# ...
    "docker": {
      "example.com": [
        {
          "type": "sigstoreSigned",
          "keyData": "a2V5RGF0YQ==",
          "rekorPublicKeyData": "cmVrb3JLZXIEYXRh",
          "signedIdentity": {
            "type": "matchRepoDigestOrExact"
          }
        }
      ],
# ...
```

Example output for the cluster image policy object for a BYOPKI certificate showing the new cluster image policy

```
# ...
  "transports": {
# ...
    "docker": {
      "example.io": [
        {
          "type": "sigstoreSigned",
          "pki": {
            "caRootsData":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk...URS0tLS0t",
            "caIntermediatesData":
"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1J....IDQVRFLS0tLS0=",
            "subjectEmail": "email@example.com",
            "subjectHostname": "myhost.example.com"
          },
          "signedIdentity": {
            "type": "matchRepository"
          }
        }
      ],
    },
  ],

```

Example output for the cluster image policy object with a Fulcio certificate showing the new cluster image policy

```
# ...
  "transports": {
# ...
    "docker": {
      "example.com": [
        {
          "type": "sigstoreSigned",
          "fulcio": {
            "caData": "a2V5RGF0YQ==",
            "oidcIssuer": "https://expected.OIDC.issuer/",
            "subjectEmail": "expected-signing-user@example.com"
          },
          "rekorPublicKeyData": "cmVrb3JLZXIEYXRh",
          "signedIdentity": {
            "type": "remapIdentity",
            "prefix": "example.com",
            "signedPrefix": "mirror-example.com"
          }
        }
      ],
    },
  ],
# ...

```

- d. Examine the **sigstore-registries.yaml** file by running the following command:

```
sh-5.1# cat /etc/containers/registries.d/sigstore-registries.yaml
```

Example output showing that the scoped registry was added

```
docker:
  example.com:
```

```
use-sigstore-attachments: true 1
quay.io/openshift-release-dev/ocp-release:
  use-sigstore-attachments: true
```

- ¹ When **true**, specifies that sigstore signatures are going to be read along with the image.

Additional resources

- [About cluster and image policy parameters](#)

12.4. CREATING AN IMAGE POLICY CR

An **ImagePolicy** custom resource (CR) enables a cluster administrator or application developer to configure a sigstore signature verification policy for a specific namespace. The MCO watches **ImagePolicy** instances in different namespaces and updates the `/etc/crio/policies/<namespace>.json` and `/etc/containers/registries.d/sigstore-registries.yaml` files on all the nodes in the cluster.



NOTE

If a scoped image or repository in an image policy is nested under one of the scoped images or repositories in a cluster image policy, only the policy from cluster image policy is applied. However, the image policy object is created with an error message. For example, if an image policy specifies **example.com/global/image**, and the cluster image policy specifies **example.com/global**, the namespace inherits the policy from the cluster image policy.

The following example shows general guidelines on how to configure an **ImagePolicy** object. For more details on the parameters, see "About cluster and image policy parameters".

Prerequisites

- You have a sigstore-supported public key infrastructure (PKI) key, a Bring Your Own Public Key Infrastructure (BYOPKI) certificate, or provide a [Cosign public and private key pair](#) for signing operations.
- You have a signing process in place to sign your images.
- You have access to a registry that supports Cosign signatures, if you are using Cosign signatures.
- If you are using a BYOPKI certificate as the root of trust, you enabled the required Technology Preview features for your cluster by editing the **FeatureGate** CR named **cluster**:

```
$ oc edit featuregate cluster
```

Example FeatureGate CR

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
```

```
name: cluster
spec:
  featureSet: TechPreviewNoUpgrade 1
```

- 1** Enables the required **SigstoreImageVerification** feature.



WARNING

Enabling the **TechPreviewNoUpgrade** feature set on your cluster cannot be undone and prevents minor version updates. This feature set allows you to enable these Technology Preview features on test clusters, where you can fully test them. Do not enable this feature set on production clusters.

Procedure

1. Create an image policy object similar to the following examples. See "About cluster and image policy parameters" for specific details on these parameters.

Example image policy object with a public key policy and the **MatchRepository** match policy

```
apiVersion: config.openshift.io/v1
kind: ImagePolicy 1
metadata:
  name: p0
  namespace: mynamespace 2
spec:
  scopes: 3
  - example.io/crio/signed
  policy: 4
  rootOfTrust: 5
  policyType: PublicKey 6
  publicKey:
    keyData: a2V5RGF0YQ== 7
    rekorKeyData: cmVrb3JLZXIEYXRh 8
  signedIdentity:
    matchPolicy: MatchRepository 9
```

- 1** Creates an **ImagePolicy** object.
- 2** Specifies the namespace where the image policy is applied.
- 3** Defines a list of repositories or images assigned to this policy.
- 4** Specifies the parameters that define how the images are verified.
- 5** Defines a root of trust for the policy.

- 6 Specifies the policy types that define the root of trust, either a public key, a BYOPKI certificate, or a [Fulcio certificate](#). Here, a public key with Rekor verification.
- 7 For a public key policy, specifies a base64-encoded public key in the PEM format. The maximum length is 8192 characters.
- 8 Optional: Specifies a base64-encoded Rekor public key in the PEM format. The maximum length is 8192 characters.
- 9 Optional: Specifies one of the following processes to verify the identity in the signature and the actual image identity:
 - **MatchRepoDigestOrExact**.
 - **MatchRepository**.
 - **ExactRepository**. The **exactRepository** parameter must be specified.
 - **RemapIdentity**. The **prefix** and **signedPrefix** parameters must be specified.

Example image policy object for a BYOPKI policy and the **MatchRepository** match policy

```

apiVersion: config.openshift.io/v1alpha1
kind: ImagePolicy 1
metadata:
  name: pki-policy
  namespace: mynamespace 2
spec:
  scopes:
  - example.io 3
  policy: 4
  rootOfTrust: 5
  policyType: PKI 6
  pki: 7
    caRootsData:
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk....RVJUSUZJQ0FURS0tLS0t
    caIntermediatesData:
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURkVENDQ....0QT09Ci0tLS0tRU5EIE
    NFUIRJRkIDQVRFLS0tLS0=
    pkiCertificateSubject: 8
      email: email@example.com
      hostname: myhost.example.com
  signedIdentity:
    matchPolicy: MatchRepository 9

```

- 1 Creates an **ImagePolicy** object.
- 2 Specifies the namespace where the image policy is applied.
- 3 Defines a list of repositories or images assigned to this policy.
- 4 Specifies the parameters that define how the images are verified.

- 5 Defines a root of trust for the policy.
- 6 Specifies the policy types that define the root of trust, either a public key, a BYOPKI certificate, or a [Fulcio certificate](#). Here, a BYOPKI certificate.
- 7 For a BYOPKI certificate, specify **caRootsData**. This parameter specifies a base64-encoded CA root certificate in the PEM format. The maximum length is 8192 characters. Optionally with **caIntermediatesData**, specifies a base64-encoded intermediate CA root certificate in the PEM format. The maximum length is 8192 characters.
- 8 Specifies a subject alternative name (SAN) to authenticate the user's identity by using a hostname and an email address:
 - **email**. Specifies the email address specified when the certificate was generated.
 - **hostname**. Specifies the hostname specified when the certificate was generated.
- 9 For a BYOPKI certificate, specify **MatchRepository** to verify the identity in the signature and the actual image identity. The default signed identity is **matchRepoDigestOrExact**, which requires digest specification. The signature in this case was not created for digested image.

Example image policy object with a Fulcio certificate policy and the **ExactRepository** match policy

```

apiVersion: config.openshift.io/v1
kind: ImagePolicy 1
metadata:
  name: p1
  namespace: mynamespace 2
spec:
  scopes: 3
  - example.io/crio/signed
  policy: 4
  rootOfTrust: 5
  policyType: FulcioCAWithRekor 6
  fulcioCAWithRekor: 7
    fulcioCAData: a2V5RGF0YQ==
    fulcioSubject:
      oidcIssuer: "https://expected.OIDC.issuer/"
      signedEmail: "expected-signing-user@example.com"
    rekorKeyData: cmVrb3JLZXIEYXRh 8
  signedIdentity:
    matchPolicy: ExactRepository 9
    exactRepository:
      repository: quay.io/crio/signed 10

```

- 1 Creates an **ImagePolicy** object.
- 2 Specifies the namespace where the image policy is applied.
- 3 Defines a list of repositories or images assigned to this policy.

- 4 Specifies the parameters that define how the images are verified.
- 5 Defines a root of trust for the policy.
- 6 Specifies the policy types that define the root of trust, either a public key, a BYOPKI certificate, or a [Fulcio certificate](#). Here, a Fulcio certificate with required Rekor verification.
- 7 For a Fulcio certificate policy, the following parameters are required:
 - **fulcioCAData**: Specifies a base64-encoded Fulcio certificate in the PEM format. The maximum length is 8192 characters.
 - **fulcioSubject**: Specifies the OIDC issuer and the email of the Fulcio authentication configuration.
- 8 Specifies a base64-encoded Rekor public key in the PEM format. This parameter is required when the **policyType** is **FulcioCAWithRekor**. The maximum length is 8192 characters.
- 9 Optional: Specifies one of the following processes to verify the identity in the signature and the actual image identity:
 - **MatchRepoDigestOrExact**.
 - **MatchRepository**.
 - **ExactRepository**. The **exactRepository** parameter must be specified.
 - **RemapIdentity**. The **prefix** and **signedPrefix** parameters must be specified.
- 10 For the **exactRepository** match policy, specifies the repository that contains the image identity and signature.

2. Create the image policy object:

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

The Machine Config Operator (MCO) updates the machine config pools (MCP) in your cluster.

Verification

- After the nodes in your cluster are updated, you can verify that the image policy has been configured:
 - a. Start a debug pod for the node by running the following command:


```
$ oc debug node/<node_name>
```
 - b. Set **/host** as the root directory within the debug shell by running the following command:


```
sh-5.1# chroot /host/
```
 - c. Examine the **<namespace>.json** file by running the following command:

```
sh-5.1# cat /etc/crio/policies/<namespace>.json
```

Example output for the image policy object with a public key showing the new image policy

```
# ...
"transports": {
# ...
  "docker": {
    "example.io/crio/signed": [
      {
        "type": "sigstoreSigned",
        "keyData": "a2V5RGF0YQ==",
        "rekorPublicKeyData": "cmVrb3JLZXIEYXRh",
        "signedIdentity": {
          "type": "matchRepository",
          "dockerRepository": "example.org/crio/signed"
        }
      }
    ]
  }
# ...
```

Example output for the image policy object for a BYOPKI certificate showing the new image policy

```
# ...
"transports": {
# ...
  "docker": {
    "docker.io": [
      {
        "type": "sigstoreSigned",
        "pki": {
          "caRootsData": "LS0t...LS0t",
          "caIntermediatesData": "LS0t...LS0t",
          "subjectEmail": "email@example.com",
          "subjectHostname": "myhost.example.com"
        },
        "signedIdentity": {
          "type": "matchRepository"
        }
      }
    ]
  },
],
```

Example output for the image policy object with a Fulcio certificate showing the new image policy

```
# ...
"transports": {
# ...
  "docker": {
    "example.io/crio/signed": [
      {
        "type": "sigstoreSigned",
        "fulcio": {
          "caData": "a2V5RGF0YQ==",
```

```

    "oidcIssuer": "https://expected.OIDC.issuer/",
    "subjectEmail": "expected-signing-user@example.com"
  },
  "rekorPublicKeyData": "cmVrb3JLZXIEYXRh",
  "signedIdentity": {
    "type": "exactRepository",
    "dockerRepository": "quay.io/crio/signed"
  }
},
# ...

```

- d. Examine the **sigstore-registries.yaml** file by running the following command:

```
sh-5.1# cat /etc/containers/registries.d/sigstore-registries.yaml
```

Example output showing that the scoped registry was added

```

docker:
  example.io/crio/signed:
    use-sigstore-attachments: true 1
  quay.io/openshift-release-dev/ocp-release:
    use-sigstore-attachments: true

```

- 1** When **true**, specifies that sigstore signatures are going to be read along with the image.

- e. Check the crio log for sigstore signature verification by running the following command:

```
sh-5.1# journalctl -u crio | grep -A 100 "Pulling image: example.io/crio"
```

Example output with timestamp removed

```

# ...
msg="IsRunningImageAllowed for image docker:example.io/crio/signed:latest"
file="signature/policy_eval.go:274" 1
msg="Using transport \"docker\" specific policy section \"example.io/crio/signed\""
file="signature/policy_eval.go:150" 2
msg="Reading
/var/lib/containers/sigstore/crio/signed@sha256=18b42e8ea347780f35d979a829affa17859
3a8e31d90644466396e1187a07f3a/signature-1" file="docker/docker_image_src.go:545"
msg="Looking for Sigstore attachments in quay.io/crio/signed:sha256-
18b42e8ea347780f35d979a829affa178593a8e31d90644466396e1187a07f3a.sig"
file="docker/docker_client.go:1138"
msg="GET https://quay.io/v2/crio/signed/manifests/sha256-
18b42e8ea347780f35d979a829affa178593a8e31d90644466396e1187a07f3a.sig"
file="docker/docker_client.go:617"
msg="Content-Type from manifest GET is
\"application/vnd.oci.image.manifest.v1+json\"" file="docker/docker_client.go:989"
msg="Found a Sigstore attachment manifest with 1 layers"
file="docker/docker_image_src.go:639"
msg="Fetching Sigstore attachment 1/1:

```

```
sha256:8276724a208087e73ae5d9d6e8f872f67808c08b0acdfdc73019278807197c45"  
file="docker/docker_image_src.go:644"  
# ...
```

- 1 The **IsRunningImageAllowed** line confirms that image is allowed by the configured sigstore verification policy.
- 2 The **Using transport \"docker\" specific policy section \"example.io/crio/signed\"** **file=\"signature/policy_eval.go:150** line confirms that the image policy has been applied.

Additional resources

- [About cluster and image policy parameters](#)