



OpenShift Container Platform 4.18

Scalability and performance

Scaling your OpenShift Container Platform cluster and tuning performance in production environments

OpenShift Container Platform 4.18 Scalability and performance

Scaling your OpenShift Container Platform cluster and tuning performance in production environments

Legal Notice

Copyright © 2025 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

This document provides instructions for scaling your cluster and optimizing the performance of your OpenShift Container Platform environment.

Table of Contents

CHAPTER 1. OPENSOURCE CONTAINER PLATFORM SCALABILITY AND PERFORMANCE OVERVIEW	11
Recommended performance and scalability practices	11
Telco reference design specifications	11
Planning, optimization, and measurement	11
CHAPTER 2. RECOMMENDED PERFORMANCE AND SCALABILITY PRACTICES	12
2.1. RECOMMENDED CONTROL PLANE PRACTICES	12
2.1.1. Recommended practices for scaling the cluster	12
2.1.2. Control plane node sizing	12
2.1.2.1. Selecting a larger Amazon Web Services instance type for control plane machines	15
2.1.2.1.1. Changing the Amazon Web Services instance type by using a control plane machine set	15
2.1.2.1.2. Changing the Amazon Web Services instance type by using the AWS console	16
2.1.3. Recommended practices for scaling the Cluster Monitoring Operator	17
2.1.4. Recommended practices for scaling the Prometheus database	18
2.1.5. Recommended practices for scaling the Configuring cluster monitoring	19
2.1.6. Recommended practices for scaling the Additional resources	20
2.2. RECOMMENDED INFRASTRUCTURE PRACTICES	17
2.2.1. Infrastructure node sizing	17
2.2.2. Scaling the Cluster Monitoring Operator	18
2.2.3. Prometheus database storage requirements	18
2.2.4. Configuring cluster monitoring	19
2.2.5. Additional resources	20
2.3. RECOMMENDED ETCD PRACTICES	20
2.3.1. Storage practices for etcd	20
2.3.2. Validating the hardware for etcd	21
2.3.3. Node scaling for etcd	22
2.3.4. Moving etcd to a different disk	23
2.3.5. Defragmenting etcd data	26
2.3.5.1. Automatic defragmentation	27
2.3.5.2. Manual defragmentation	28
2.3.6. Setting tuning parameters for etcd	30
2.3.6.1. Changing hardware speed tolerance	30
2.3.7. Increasing the database size for etcd	32
2.3.7.1. Changing the etcd database size	33
2.3.7.2. Troubleshooting	34
2.3.7.2.1. Value is too small	35
2.3.7.2.2. Value is too large	35
2.3.7.2.3. Value is decreasing	35
CHAPTER 3. TELCO CORE REFERENCE DESIGN SPECIFICATIONS	37
3.1. TELCO CORE RDS 4.18 USE MODEL OVERVIEW	37
3.2. ABOUT THE TELCO CORE CLUSTER USE MODEL	37
3.3. REFERENCE DESIGN SCOPE	38
3.4. DEVIATIONS FROM THE REFERENCE DESIGN	38
3.5. TELCO CORE COMMON BASELINE MODEL	39
3.6. TELCO CORE CLUSTER COMMON USE MODEL ENGINEERING CONSIDERATIONS	40
3.6.1. Application workloads	41
3.6.2. Signaling workloads	42
3.7. TELCO CORE RDS COMPONENTS	42
3.7.1. CPU partitioning and performance tuning	42
3.7.2. Service mesh	43
3.7.3. Networking	44
3.7.3.1. Cluster Network Operator	46
3.7.3.2. Load balancer	47
3.7.3.3. SR-IOV	48

3.7.3.4. NMState Operator	49
3.7.4. Logging	50
3.7.5. Power Management	50
3.7.6. Storage	51
3.7.6.1. Red Hat OpenShift Data Foundation	51
3.7.6.2. Additional storage solutions	51
3.7.7. Telco core deployment components	52
3.7.7.1. Red Hat Advanced Cluster Management	52
3.7.7.2. Topology Aware Lifecycle Manager	52
3.7.7.3. GitOps Operator and GitOps ZTP plugins	53
3.7.7.4. Monitoring	54
3.7.8. Scheduling	55
3.7.9. Node Configuration	56
3.7.10. Host firmware and boot loader configuration	57
3.7.11. Disconnected environment	58
3.7.12. Agent-based Installer	58
3.7.13. Security	59
3.7.14. Scalability	61
3.8. TELCO CORE REFERENCE CONFIGURATION CRS	61
3.8.1. Extracting the telco core reference design configuration CRs	61
3.8.2. Comparing a cluster with the telco core reference configuration	62
3.8.3. Node configuration reference CRs	65
3.8.4. Resource tuning reference CRs	66
3.8.5. Networking reference CRs	66
3.8.6. Scheduling reference CRs	70
3.8.7. Storage reference CRs	70
3.9. TELCO CORE REFERENCE CONFIGURATION SOFTWARE SPECIFICATIONS	71
CHAPTER 4. TELCO RAN DU REFERENCE DESIGN SPECIFICATIONS	73
4.1. REFERENCE DESIGN SPECIFICATIONS FOR TELCO RAN DU 5G DEPLOYMENTS	73
4.2. REFERENCE DESIGN SCOPE	74
4.3. DEVIATIONS FROM THE REFERENCE DESIGN	75
4.4. ENGINEERING CONSIDERATIONS FOR THE RAN DU USE MODEL	75
4.5. TELCO RAN DU APPLICATION WORKLOADS	77
4.6. TELCO RAN DU REFERENCE DESIGN COMPONENTS	78
4.6.1. Host firmware tuning	79
4.6.2. CPU partitioning and performance tuning	80
4.6.3. PTP Operator	81
4.6.4. SR-IOV Operator	81
4.6.5. Logging	83
4.6.6. SRIOV-FEC Operator	83
4.6.7. Lifecycle Agent	83
4.6.8. Local Storage Operator	84
4.6.9. Logical Volume Manager Storage	84
4.6.10. Workload partitioning	85
4.6.11. Cluster tuning	85
4.6.12. Machine configuration	87
4.7. TELCO RAN DU DEPLOYMENT COMPONENTS	88
4.7.1. Red Hat Advanced Cluster Management	88
4.7.2. SiteConfig Operator	89
4.7.3. Topology Aware Lifecycle Manager	90
4.7.4. GitOps Operator and GitOps ZTP	91
4.7.5. Agent-based Installer	92

4.8. TELCO RAN DU REFERENCE CONFIGURATION CRS	93
4.8.1. Cluster tuning reference CRs	93
4.8.2. Day 2 Operators reference CRs	94
4.8.3. Machine configuration reference CRs	101
4.9. COMPARING A CLUSTER WITH THE TELCO RAN DU REFERENCE CONFIGURATION	102
4.10. TELCO RAN DU 4.18 VALIDATED SOFTWARE COMPONENTS	105
4.11. TELCO RAN DU 4.18 HUB CLUSTER VALIDATED SOFTWARE COMPONENTS	105
CHAPTER 5. TELCO HUB REFERENCE DESIGN SPECIFICATIONS	107
5.1. REFERENCE DESIGN SCOPE	107
5.2. DEVIATIONS FROM THE REFERENCE DESIGN	107
5.3. HUB CLUSTER ARCHITECTURE OVERVIEW	108
5.4. TELCO MANAGEMENT HUB CLUSTER USE MODEL	110
5.5. HUB CLUSTER SCALING TARGET	111
5.6. HUB CLUSTER RESOURCE UTILIZATION	111
5.7. HUB CLUSTER TOPOLOGY	112
5.8. HUB CLUSTER NETWORKING	113
5.9. HUB CLUSTER MEMORY AND CPU REQUIREMENTS	114
5.10. HUB CLUSTER STORAGE REQUIREMENTS	114
5.10.1. Assisted Service	115
5.10.2. RHACM Observability	115
5.10.3. Storage considerations	116
5.10.4. Git repository	117
5.11. OPENSHIFT CONTAINER PLATFORM INSTALLATION ON THE HUB CLUSTER	118
5.12. DAY 2 OPERATORS IN THE HUB CLUSTER	119
5.13. OBSERVABILITY	120
5.14. MANAGED CLUSTER LIFECYCLE MANAGEMENT	122
5.14.1. Managed cluster deployment	122
5.14.2. Managed cluster updates	122
5.15. HUB CLUSTER DISASTER RECOVERY	124
5.16. HUB CLUSTER COMPONENTS	124
5.16.1. Red Hat Advanced Cluster Management (RHACM)	124
5.16.2. Topology Aware Lifecycle Manager	126
5.16.3. GitOps Operator and GitOps ZTP	127
5.16.4. Local Storage Operator	128
5.16.5. Red Hat OpenShift Data Foundation	128
5.16.6. Logging	129
5.16.7. OpenShift API for Data Protection	129
5.17. HUB CLUSTER REFERENCE CONFIGURATION CRS	130
5.17.1. RHACM reference YAML	130
5.17.2. Storage reference YAML	140
5.17.3. GitOps Operator and GitOps ZTP reference YAML	143
5.17.4. Logging reference YAML	150
5.17.5. Installation reference YAML	151
5.18. TELCO HUB REFERENCE CONFIGURATION SOFTWARE SPECIFICATIONS	154
CHAPTER 6. COMPARING CLUSTER CONFIGURATIONS	156
6.1. UNDERSTANDING THE CLUSTER-COMPARE PLUGIN	156
6.1.1. Overview of the cluster-compare plugin	156
6.1.2. Understanding a reference configuration	157
6.1.3. Additional resources	158
6.2. INSTALLING THE CLUSTER-COMPARE PLUGIN	158
6.2.1. Installing the cluster-compare plugin	158

6.2.2. Additional resources	159
6.3. USING THE CLUSTER-COMPARE PLUGIN	159
6.3.1. Using the cluster-compare plugin with a live cluster	159
6.3.2. Using the cluster-compare plugin with must-gather data	161
6.3.3. Reference cluster-compare plugin options	163
6.3.4. Example: Comparing a cluster with the telco core reference configuration	164
6.3.5. Additional resources	168
6.4. CREATING A REFERENCE CONFIGURATION	168
6.4.1. Structure of the metadata.yaml file	168
6.4.2. Configuring template relationships	168
6.4.3. Configuring expected variation in a template	169
6.4.4. Configuring the metadata.yaml file to exclude template fields	170
6.4.4.1. Excluding all fields not specified in a template	171
6.4.4.2. Excluding specific fields by setting default exclusion fields	171
6.4.4.3. Excluding specific fields	172
6.4.4.4. Excluding specific fields by setting default exclusion groups	172
6.4.5. Configuring inline validation for template fields	173
6.4.5.1. Configuring inline validation with the regex function	173
6.4.5.2. Configuring inline validation with the capturegroups function	174
6.4.6. Configuring descriptions for the output	175
6.5. PERFORMING ADVANCED REFERENCE CONFIGURATION CUSTOMIZATION	175
6.5.1. Configuring manual matching between CRs and templates	176
6.5.2. Patching a reference configuration	177
6.5.2.1. Using the cluster-compare plugin to generate a patch	177
6.5.2.2. Creating a patch file manually	179
6.6. TROUBLESHOOTING CLUSTER COMPARISONS	181
6.6.1. Troubleshooting false positives for missing resources	181
6.6.2. Troubleshooting multiple template matches for the same CR	182
6.6.3. Additional resources	182
CHAPTER 7. PLANNING YOUR ENVIRONMENT ACCORDING TO OBJECT MAXIMUMS	183
7.1. OPENSPLIT CONTAINER PLATFORM TESTED CLUSTER MAXIMUMS FOR MAJOR RELEASES	183
7.1.1. Example scenario	185
7.2. OPENSPLIT CONTAINER PLATFORM ENVIRONMENT AND CONFIGURATION ON WHICH THE CLUSTER MAXIMUMS ARE TESTED	186
7.2.1. AWS cloud platform	186
7.2.2. IBM Power platform	187
7.2.3. IBM Z platform	187
7.3. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO TESTED CLUSTER MAXIMUMS	188
7.4. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO APPLICATION REQUIREMENTS	189
CHAPTER 8. USING QUOTAS AND LIMIT RANGES	192
8.1. RESOURCES MANAGED BY QUOTA	192
8.1.1. Setting resource quota for extended resources	194
8.1.2. Quota scopes	197
8.2. ADMIN QUOTA USAGE	198
8.2.1. Quota enforcement	198
8.2.2. Requests compared to limits	198
8.2.3. Sample resource quota definitions	198
8.2.4. Creating a quota	202
8.2.5. Creating object count quotas	202
8.2.6. Viewing a quota	203
8.2.7. Configuring quota synchronization period	203

8.2.8. Explicit quota to consume a resource	204
8.3. SETTING LIMIT RANGES	205
8.3.1. Container limits	207
8.3.2. Pod limits	208
8.3.3. Image limits	209
8.3.4. Image stream limits	209
8.3.5. Counting of image references	210
8.3.6. PersistentVolumeClaim limits	210
8.4. LIMIT RANGE OPERATIONS	211
8.4.1. Creating a limit range	211
8.4.2. View the limit	211
8.4.3. Deleting a limit range	212
CHAPTER 9. RECOMMENDED HOST PRACTICES FOR IBM Z & IBM LINUXONE ENVIRONMENTS	213
9.1. MANAGING CPU OVERCOMMITMENT	213
9.2. DISABLE TRANSPARENT HUGE PAGES	213
9.3. BOOST NETWORKING PERFORMANCE WITH RECEIVE FLOW STEERING	214
9.3.1. Use the Machine Config Operator (MCO) to activate RFS	214
9.4. CHOOSE YOUR NETWORKING SETUP	215
9.5. ENSURE HIGH DISK PERFORMANCE WITH HYPERPAV ON Z/VM	215
9.5.1. Use the Machine Config Operator (MCO) to activate HyperPAV aliases in nodes using z/VM full-pack minidisks	216
9.6. RHEL KVM ON IBM Z HOST RECOMMENDATIONS	217
9.6.1. Use I/O threads for your virtual block devices	217
9.6.2. Avoid virtual SCSI devices	218
9.6.3. Configure guest caching for disk	218
9.6.4. Exclude the memory balloon device	218
9.6.5. Tune the CPU migration algorithm of the host scheduler	218
9.6.6. Disable the cpuset cgroup controller	219
9.6.7. Tune the polling period for idle virtual CPUs	219
CHAPTER 10. USING THE NODE TUNING OPERATOR	221
10.1. ABOUT THE NODE TUNING OPERATOR	221
10.2. ACCESSING AN EXAMPLE NODE TUNING OPERATOR SPECIFICATION	221
10.3. DEFAULT PROFILES SET ON A CLUSTER	222
10.4. VERIFYING THAT THE TUNED PROFILES ARE APPLIED	222
10.5. CUSTOM TUNING SPECIFICATION	223
10.6. CUSTOM TUNING EXAMPLES	228
10.7. DEFERRING APPLICATION OF TUNING CHANGES	230
10.7.1. Deferring application of tuning changes: An example	232
10.8. SUPPORTED TUNED DAEMON PLUGINS	235
10.9. CONFIGURING NODE TUNING IN A HOSTED CLUSTER	236
10.10. ADVANCED NODE TUNING FOR HOSTED CLUSTERS BY SETTING KERNEL BOOT PARAMETERS	239
CHAPTER 11. USING CPU MANAGER AND TOPOLOGY MANAGER	242
11.1. SETTING UP CPU MANAGER	242
11.2. TOPOLOGY MANAGER POLICIES	247
11.3. SETTING UP TOPOLOGY MANAGER	248
11.4. POD INTERACTIONS WITH TOPOLOGY MANAGER POLICIES	248
CHAPTER 12. SCHEDULING NUMA-AWARE WORKLOADS	250
12.1. ABOUT NUMA-AWARE SCHEDULING	250
Introduction to NUMA	250
Performance considerations	250

NUMA-aware scheduling	250
Integration with Node Tuning Operator	250
Default scheduling logic	250
NUMA-aware pod scheduling diagram	250
12.2. NUMA RESOURCE SCHEDULING STRATEGIES	251
Scoring strategy summary	252
LeastAllocated strategy example	252
MostAllocated strategy example	253
BalancedAllocation strategy example	253
12.3. INSTALLING THE NUMA RESOURCES OPERATOR	253
12.3.1. Installing the NUMA Resources Operator using the CLI	254
12.3.2. Installing the NUMA Resources Operator using the web console	255
12.4. SCHEDULING NUMA-AWARE WORKLOADS	256
12.4.1. Creating the NUMAResourcesOperator custom resource	256
12.4.2. Deploying the NUMA-aware secondary pod scheduler	258
12.4.3. Configuring a single NUMA node policy	259
12.4.4. Sample performance profile	259
12.4.5. Creating a KubeletConfig CR	260
12.4.6. Scheduling workloads with the NUMA-aware scheduler	261
12.5. OPTIONAL: CONFIGURING POLLING OPERATIONS FOR NUMA RESOURCES UPDATES	264
12.6. TROUBLESHOOTING NUMA-AWARE SCHEDULING	266
12.6.1. Reporting more exact resource availability	269
12.6.2. Changing where high-performance workloads run	272
12.6.3. Checking the NUMA-aware scheduler logs	273
12.6.4. Troubleshooting the resource topology exporter	276
12.6.5. Correcting a missing resource topology exporter config map	277
12.6.6. Collecting NUMA Resources Operator data	279
CHAPTER 13. SCALABILITY AND PERFORMANCE OPTIMIZATION	280
13.1. OPTIMIZING STORAGE	280
13.1.1. Available persistent storage options	280
13.1.2. Recommended configurable storage technology	281
13.1.2.1. Specific application storage recommendations	282
13.1.2.1.1. Registry	282
13.1.2.1.2. Scaled registry	282
13.1.2.1.3. Metrics	283
13.1.2.1.4. Logging	283
13.1.2.1.5. Applications	283
13.1.2.2. Other specific application storage recommendations	283
13.1.3. Data storage management	284
13.1.4. Optimizing storage performance for Microsoft Azure	285
13.2. OPTIMIZING ROUTING	285
13.2.1. Baseline Ingress Controller (router) performance	285
13.2.2. Configuring Ingress Controller liveness, readiness, and startup probes	287
13.2.3. Configuring HAProxy reload interval	288
13.3. OPTIMIZING NETWORKING	289
13.3.1. Optimizing the MTU for your network	289
13.3.2. Recommended practices for installing large scale clusters	290
13.3.3. Impact of IPsec	290
13.3.4. Additional resources	290
13.4. OPTIMIZING CPU USAGE WITH MOUNT NAMESPACE ENCAPSULATION	290
13.4.1. Encapsulating mount namespaces	291
13.4.2. Configuring mount namespace encapsulation	293

13.4.3. Inspecting encapsulated namespaces	296
13.4.4. Running additional services in the encapsulated namespace	297
13.4.5. Additional resources	297
CHAPTER 14. MANAGING BARE-METAL HOSTS	298
14.1. ABOUT BARE METAL HOSTS AND NODES	298
14.2. MAINTAINING BARE METAL HOSTS	298
14.2.1. Adding a bare metal host to the cluster using the web console	298
14.2.2. Adding a bare metal host to the cluster using YAML in the web console	299
14.2.3. Automatically scaling machines to the number of available bare metal hosts	300
14.2.4. Removing bare metal hosts from the provisioner node	301
14.2.5. Powering off bare-metal hosts	302
CHAPTER 15. WHAT HUGE PAGES DO AND HOW THEY ARE CONSUMED BY APPLICATIONS	304
15.1. WHAT HUGE PAGES DO	304
15.2. HOW HUGE PAGES ARE CONSUMED BY APPS	304
15.3. CONSUMING HUGE PAGES RESOURCES USING THE DOWNWARD API	305
15.4. CONFIGURING HUGE PAGES AT BOOT TIME	307
15.5. DISABLING TRANSPARENT HUGE PAGES	309
CHAPTER 16. UNDERSTANDING LOW LATENCY TUNING FOR CLUSTER NODES	310
16.1. ABOUT LOW LATENCY	310
16.2. ABOUT HYPER-THREADING FOR LOW LATENCY AND REAL-TIME APPLICATIONS	311
CHAPTER 17. TUNING NODES FOR LOW LATENCY WITH THE PERFORMANCE PROFILE	312
17.1. CREATING A PERFORMANCE PROFILE	312
17.1.1. About the Performance Profile Creator	312
17.1.2. Creating a machine config pool to target nodes for performance tuning	313
17.1.3. Gathering data about your cluster for the PPC	314
17.1.4. Running the Performance Profile Creator using Podman	315
17.1.5. Running the Performance Profile Creator wrapper script	319
17.1.6. Performance Profile Creator arguments	324
17.1.7. Reference performance profiles	327
17.1.7.1. Performance profile template for clusters that use OVS-DPDK on OpenStack	327
17.1.7.2. Telco RAN DU reference design performance profile	328
17.1.7.3. Telco core reference design performance profile	329
17.2. SUPPORTED PERFORMANCE PROFILE API VERSIONS	330
Upgrading the performance profile to use device interrupt processing	330
Upgrading Node Tuning Operator API from v1alpha1 to v1	330
Upgrading Node Tuning Operator API from v1alpha1 or v1 to v2	330
17.3. CONFIGURING NODE POWER CONSUMPTION AND REALTIME PROCESSING WITH WORKLOAD HINTS	330
17.4. CONFIGURING POWER SAVING FOR NODES THAT RUN COLOCATED HIGH AND LOW PRIORITY WORKLOADS	332
17.5. RESTRICTING CPUS FOR INFRA AND APPLICATION CONTAINERS	333
17.6. CONFIGURING HYPER-THREADING FOR A CLUSTER	335
17.6.1. Disabling Hyper-Threading for low latency applications	337
17.7. MANAGING DEVICE INTERRUPT PROCESSING FOR GUARANTEED POD ISOLATED CPUS	338
17.7.1. Finding the effective IRQ affinity setting for a node	338
17.7.2. Configuring node interrupt affinity	339
17.8. CONFIGURING HUGE PAGES	340
17.8.1. Allocating multiple huge page sizes	341
17.9. REDUCING NIC QUEUES USING THE NODE TUNING OPERATOR	341
17.9.1. Adjusting the NIC queues with the performance profile	341

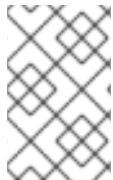
17.9.2. Verifying the queue status	345
17.9.3. Logging associated with adjusting NIC queues	348
CHAPTER 18. PROVISIONING REAL-TIME AND LOW LATENCY WORKLOADS	349
18.1. SCHEDULING A LOW LATENCY WORKLOAD ONTO A WORKER WITH REAL-TIME CAPABILITIES	349
18.2. CREATING A POD WITH A GUARANTEED QOS CLASS	352
18.3. DISABLING CPU LOAD BALANCING IN A POD	354
18.4. DISABLING POWER SAVING MODE FOR HIGH PRIORITY PODS	355
18.5. DISABLING CPU CFS QUOTA	356
18.6. DISABLING INTERRUPT PROCESSING FOR CPUS WHERE PINNED CONTAINERS ARE RUNNING	357
CHAPTER 19. DEBUGGING LOW LATENCY NODE TUNING STATUS	358
19.1. DEBUGGING LOW LATENCY CNF TUNING STATUS	358
19.1.1. Machine config pools	359
19.2. COLLECTING LOW LATENCY TUNING DEBUGGING DATA FOR RED HAT SUPPORT	360
19.2.1. About the must-gather tool	360
19.2.2. Gathering low latency tuning data	361
CHAPTER 20. PERFORMING LATENCY TESTS FOR PLATFORM VERIFICATION	363
20.1. PREREQUISITES FOR RUNNING LATENCY TESTS	363
20.2. MEASURING LATENCY	363
20.3. RUNNING THE LATENCY TESTS	364
20.3.1. Running hwlatdetect	366
Example hwlatdetect test results	368
20.3.2. Running cyclictest	369
Example cyclictest results	371
20.3.3. Running oslat	372
20.4. GENERATING A LATENCY TEST FAILURE REPORT	374
20.5. GENERATING A JUNIT LATENCY TEST REPORT	374
20.6. RUNNING LATENCY TESTS ON A SINGLE-NODE OPENSHIFT CLUSTER	375
20.7. RUNNING LATENCY TESTS IN A DISCONNECTED CLUSTER	376
Mirroring the images to a custom registry accessible from the cluster	376
Configuring the tests to consume images from a custom registry	376
Mirroring images to the cluster OpenShift image registry	377
Mirroring a different set of test images	378
20.8. TROUBLESHOOTING ERRORS WITH THE CNF-TESTS CONTAINER	378
CHAPTER 21. IMPROVING CLUSTER STABILITY IN HIGH LATENCY ENVIRONMENTS USING WORKER LATENCY PROFILES	379
21.1. UNDERSTANDING WORKER LATENCY PROFILES	379
21.2. IMPLEMENTING WORKER LATENCY PROFILES AT CLUSTER CREATION	382
21.3. USING AND CHANGING WORKER LATENCY PROFILES	383
21.4. EXAMPLE STEPS FOR DISPLAYING RESULTING VALUES OF WORKERLATENCYPROFILE	385
CHAPTER 22. WORKLOAD PARTITIONING	387
22.1. ENABLING WORKLOAD PARTITIONING	387
22.2. PERFORMANCE PROFILES AND WORKLOAD PARTITIONING	388
22.3. SAMPLE PERFORMANCE PROFILE CONFIGURATION	388
CHAPTER 23. USING THE NODE OBSERVABILITY OPERATOR	391
23.1. WORKFLOW OF THE NODE OBSERVABILITY OPERATOR	391
23.2. INSTALLING THE NODE OBSERVABILITY OPERATOR	391
23.2.1. Installing the Node Observability Operator using the CLI	391
23.2.2. Installing the Node Observability Operator using the web console	393
23.3. REQUESTING CRI-O AND KUBELET PROFILING DATA USING THE NODE OBSERVABILITY OPERATOR	

	394
23.3.1. Creating the Node Observability custom resource	394
23.3.2. Running the profiling query	395
23.4. NODE OBSERVABILITY OPERATOR SCRIPTING	397
23.4.1. Creating the Node Observability custom resource for scripting	397
23.4.2. Configuring Node Observability Operator scripting	398
23.5. ADDITIONAL RESOURCES	400

CHAPTER 1. OPENSHIFT CONTAINER PLATFORM SCALABILITY AND PERFORMANCE OVERVIEW

OpenShift Container Platform provides best practices and tools to help you optimize the performance and scale of your clusters. The following documentation provides information on recommended performance and scalability practices, reference design specifications, optimization, and low latency tuning.

To contact Red Hat support, see [Getting support](#).



NOTE

Some performance and scalability Operators have release cycles that are independent from OpenShift Container Platform release cycles. For more information, see [OpenShift Operators](#).

Recommended performance and scalability practices

[Recommended control plane practices](#)

[Recommended infrastructure practices](#)

[Recommended etcd practices](#)

Telco reference design specifications

[Telco RAN DU reference design specification for OpenShift Container Platform 4.18](#)

[Telco core reference design specification](#)

Planning, optimization, and measurement

[Planning your environment according to object maximums](#)

[Recommended practices for IBM Z and IBM LinuxONE](#)

[Using the Node Tuning Operator](#)

[Using CPU Manager and Topology Manager](#)

[Scheduling NUMA-aware workloads](#)

[Optimizing storage, routing, networking and CPU usage](#)

[Managing bare metal hosts and events](#)

[What are huge pages and how are they used by apps](#)

[Low latency tuning for improving cluster stability and partitioning workload](#)

[Improving cluster stability in high latency environments using worker latency profiles](#)

[Workload partitioning](#)

[Using the Node Observability Operator](#)

CHAPTER 2. RECOMMENDED PERFORMANCE AND SCALABILITY PRACTICES

2.1. RECOMMENDED CONTROL PLANE PRACTICES

This topic provides recommended performance and scalability practices for control planes in OpenShift Container Platform.

2.1.1. Recommended practices for scaling the cluster

The guidance in this section is only relevant for installations with cloud provider integration.

Apply the following best practices to scale the number of worker machines in your OpenShift Container Platform cluster. You scale the worker machines by increasing or decreasing the number of replicas that are defined in the worker machine set.

When scaling up the cluster to higher node counts:

- Spread nodes across all of the available zones for higher availability.
- Scale up by no more than 25 to 50 machines at once.
- Consider creating new compute machine sets in each available zone with alternative instance types of similar size to help mitigate any periodic provider capacity constraints. For example, on AWS, use m5.large and m5d.large.

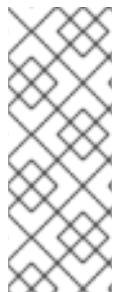


NOTE

Cloud providers might implement a quota for API services. Therefore, gradually scale the cluster.

The controller might not be able to create the machines if the replicas in the compute machine sets are set to higher numbers all at one time. The number of requests the cloud platform, which OpenShift Container Platform is deployed on top of, is able to handle impacts the process. The controller will start to query more while trying to create, check, and update the machines with the status. The cloud platform on which OpenShift Container Platform is deployed has API request limits; excessive queries might lead to machine creation failures due to cloud platform limitations.

Enable machine health checks when scaling to large node counts. In case of failures, the health checks monitor the condition and automatically repair unhealthy machines.



NOTE

When scaling large and dense clusters to lower node counts, it might take large amounts of time because the process involves draining or evicting the objects running on the nodes being terminated in parallel. Also, the client might start to throttle the requests if there are too many objects to evict. The default client queries per second (QPS) and burst rates are currently set to **50** and **100** respectively. These values cannot be modified in OpenShift Container Platform.

2.1.2. Control plane node sizing

The control plane node resource requirements depend on the number and type of nodes and objects in

the cluster. The following control plane node size recommendations are based on the results of a control plane density focused testing, or *Cluster-density*. This test creates the following objects across a given number of namespaces:

- 1 image stream
- 1 build
- 5 deployments, with 2 pod replicas in a **sleep** state, mounting 4 secrets, 4 config maps, and 1 downward API volume each
- 5 services, each one pointing to the TCP/8080 and TCP/8443 ports of one of the previous deployments
- 1 route pointing to the first of the previous services
- 10 secrets containing 2048 random string characters
- 10 config maps containing 2048 random string characters

Number of worker nodes	Cluster-density (namespaces)	CPU cores	Memory (GB)
24	500	4	16
120	1000	8	32
252	4000	16, but 24 if using the OVN-Kubernetes network plug-in	64, but 128 if using the OVN-Kubernetes network plug-in
501, but untested with the OVN-Kubernetes network plug-in	4000	16	96

The data from the table above is based on an OpenShift Container Platform running on top of AWS, using r5.4xlarge instances as control-plane nodes and m5.2xlarge instances as worker nodes.

On a large and dense cluster with three control plane nodes, the CPU and memory usage will spike up when one of the nodes is stopped, rebooted, or fails. The failures can be due to unexpected issues with power, network, underlying infrastructure, or intentional cases where the cluster is restarted after shutting it down to save costs. The remaining two control plane nodes must handle the load in order to be highly available, which leads to increase in the resource usage. This is also expected during upgrades because the control plane nodes are cordoned, drained, and rebooted serially to apply the operating system updates, as well as the control plane Operators update. To avoid cascading failures, keep the overall CPU and memory resource usage on the control plane nodes to at most 60% of all available capacity to handle the resource usage spikes. Increase the CPU and memory on the control plane nodes accordingly to avoid potential downtime due to lack of resources.



IMPORTANT

The node sizing varies depending on the number of nodes and object counts in the cluster. It also depends on whether the objects are actively being created on the cluster. During object creation, the control plane is more active in terms of resource usage compared to when the objects are in the **Running** phase.

Operator Lifecycle Manager (OLM) runs on the control plane nodes and its memory footprint depends on the number of namespaces and user installed operators that OLM needs to manage on the cluster. Control plane nodes need to be sized accordingly to avoid OOM kills. Following data points are based on the results from cluster maximums testing.

Number of namespaces	OLM memory at idle state (GB)	OLM memory with 5 user operators installed (GB)
500	0.823	1.7
1000	1.2	2.5
1500	1.7	3.2
2000	2	4.4
3000	2.7	5.6
4000	3.8	7.6
5000	4.2	9.02
6000	5.8	11.3
7000	6.6	12.9
8000	6.9	14.8
9000	8	17.7
10,000	9.9	21.6



IMPORTANT

You can modify the control plane node size in a running OpenShift Container Platform 4.18 cluster for the following configurations only:

- Clusters installed with a user-provisioned installation method.
- AWS clusters installed with an installer-provisioned infrastructure installation method.
- Clusters that use a control plane machine set to manage control plane machines.

For all other configurations, you must estimate your total node count and use the suggested control plane node size during installation.



NOTE

In OpenShift Container Platform 4.18, half of a CPU core (500 millicore) is now reserved by the system by default compared to OpenShift Container Platform 3.11 and previous versions. The sizes are determined taking that into consideration.

2.1.2.1. Selecting a larger Amazon Web Services instance type for control plane machines

If the control plane machines in an Amazon Web Services (AWS) cluster require more resources, you can select a larger AWS instance type for the control plane machines to use.



NOTE

The procedure for clusters that use a control plane machine set is different from the procedure for clusters that do not use a control plane machine set.

If you are uncertain about the state of the **ControlPlaneMachineSet** CR in your cluster, you can [verify the CR status](#).

2.1.2.1.1. Changing the Amazon Web Services instance type by using a control plane machine set

You can change the Amazon Web Services (AWS) instance type that your control plane machines use by updating the specification in the control plane machine set custom resource (CR).

Prerequisites

- Your AWS cluster uses a control plane machine set.

Procedure

1. Edit your control plane machine set CR by running the following command:

```
$ oc --namespace openshift-machine-api edit controlplanemachineset.machine.openshift.io
cluster
```

2. Edit the following line under the **providerSpec** field:

```
providerSpec:
  value:
```

...
instanceType: <compatible_aws_instance_type> ①

- ① Specify a larger AWS instance type with the same base as the previous selection. For example, you can change **m6i.xlarge** to **m6i.2xlarge** or **m6i.4xlarge**.

3. Save your changes.

- For clusters that use the default **RollingUpdate** update strategy, the Operator automatically propagates the changes to your control plane configuration.
- For clusters that are configured to use the **OnDelete** update strategy, you must replace your control plane machines manually.

Additional resources

- [Managing control plane machines with control plane machine sets](#)

2.1.2.1.2. Changing the Amazon Web Services instance type by using the AWS console

You can change the Amazon Web Services (AWS) instance type that your control plane machines use by updating the instance type in the AWS console.

Prerequisites

- You have access to the AWS console with the permissions required to modify the EC2 Instance for your cluster.
- You have access to the OpenShift Container Platform cluster as a user with the **cluster-admin** role.

Procedure

- Open the AWS console and fetch the instances for the control plane machines.
- Choose one control plane machine instance.
 - For the selected control plane machine, back up the etcd data by creating an etcd snapshot. For more information, see "Backing up etcd".
 - In the AWS console, stop the control plane machine instance.
 - Select the stopped instance, and click **Actions → Instance Settings → Change instance type**.
 - Change the instance to a larger type, ensuring that the type is the same base as the previous selection, and apply changes. For example, you can change **m6i.xlarge** to **m6i.2xlarge** or **m6i.4xlarge**.
 - Start the instance.
 - If your OpenShift Container Platform cluster has a corresponding **Machine** object for the instance, update the instance type of the object to match the instance type set in the AWS console.

3. Repeat this process for each control plane machine.

Additional resources

- [Backing up etcd](#)
- [AWS documentation about changing the instance type](#)

2.2. RECOMMENDED INFRASTRUCTURE PRACTICES

This topic provides recommended performance and scalability practices for infrastructure in OpenShift Container Platform.

2.2.1. Infrastructure node sizing

Infrastructure nodes are nodes that are labeled to run pieces of the OpenShift Container Platform environment. The infrastructure node resource requirements depend on the cluster age, nodes, and objects in the cluster, as these factors can lead to an increase in the number of metrics or time series in Prometheus. The following infrastructure node size recommendations are based on the results observed in cluster-density testing detailed in the [Control plane node sizing](#) section, where the monitoring stack and the default ingress-controller were moved to these nodes.

Number of worker nodes	Cluster density, or number of namespaces	CPU cores	Memory (GB)
27	500	4	24
120	1000	8	48
252	4000	16	128
501	4000	32	128

In general, three infrastructure nodes are recommended per cluster.



IMPORTANT

These sizing recommendations should be used as a guideline. Prometheus is a highly memory intensive application; the resource usage depends on various factors including the number of nodes, objects, the Prometheus metrics scraping interval, metrics or time series, and the age of the cluster. In addition, the router resource usage can also be affected by the number of routes and the amount/type of inbound requests.

These recommendations apply only to infrastructure nodes hosting Monitoring, Ingress and Registry infrastructure components installed during cluster creation.



NOTE

In OpenShift Container Platform 4.18, half of a CPU core (500 millicore) is now reserved by the system by default compared to OpenShift Container Platform 3.11 and previous versions. This influences the stated sizing recommendations.

2.2.2. Scaling the Cluster Monitoring Operator

OpenShift Container Platform exposes metrics that the Cluster Monitoring Operator (CMO) collects and stores in the Prometheus-based monitoring stack. As an administrator, you can view dashboards for system resources, containers, and components metrics in the OpenShift Container Platform web console by navigating to **Observe → Dashboards**.

2.2.3. Prometheus database storage requirements

Red Hat performed various tests for different scale sizes.



NOTE

- The following Prometheus storage requirements are not prescriptive and should be used as a reference. Higher resource consumption might be observed in your cluster depending on workload activity and resource density, including the number of pods, containers, routes, or other resources exposing metrics collected by Prometheus.
- You can configure the size-based data retention policy to suit your storage requirements.

Table 2.1. Prometheus Database storage requirements based on number of nodes/pods in the cluster

Number of nodes	Number of pods (2 containers per pod)	Prometheus storage growth per day	Prometheus storage growth per 15 days	Network (per tsdb chunk)
50	1800	6.3 GB	94 GB	16 MB
100	3600	13 GB	195 GB	26 MB
150	5400	19 GB	283 GB	36 MB
200	7200	25 GB	375 GB	46 MB

Approximately 20 percent of the expected size was added as overhead to ensure that the storage requirements do not exceed the calculated value.

The above calculation is for the default OpenShift Container Platform Cluster Monitoring Operator.



NOTE

CPU utilization has minor impact. The ratio is approximately 1 core out of 40 per 50 nodes and 1800 pods.

Recommendations for OpenShift Container Platform

- Use at least two infrastructure (infra) nodes.

- Use at least three **openshift-container-storage** nodes with non-volatile memory express (SSD or NVMe) drives.

2.2.4. Configuring cluster monitoring

You can increase the storage capacity for the Prometheus component in the cluster monitoring stack.

Procedure

To increase the storage capacity for Prometheus:

- 1 Create a YAML configuration file, **cluster-monitoring-config.yaml**. For example:

```
apiVersion: v1
kind: ConfigMap
data:
  config.yaml: |
    prometheusK8s:
      retention: {{PROMETHEUS_RETENTION_PERIOD}} ①
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      volumeClaimTemplate:
        spec:
          storageClassName: {{STORAGE_CLASS}} ②
          resources:
            requests:
              storage: {{PROMETHEUS_STORAGE_SIZE}} ③
      alertmanagerMain:
        nodeSelector:
          node-role.kubernetes.io/infra: ""
        volumeClaimTemplate:
          spec:
            storageClassName: {{STORAGE_CLASS}} ④
            resources:
              requests:
                storage: {{ALERTMANAGER_STORAGE_SIZE}} ⑤
    metadata:
      name: cluster-monitoring-config
      namespace: openshift-monitoring
```

① The default value of Prometheus retention is **PROMETHEUS_RETENTION_PERIOD=15d**. Units are measured in time using one of these suffixes: s, m, h, d.

② ④ The storage class for your cluster.

③ A typical value is **PROMETHEUS_STORAGE_SIZE=2000Gi**. Storage values can be a plain integer or a fixed-point integer using one of these suffixes: E, P, T, G, M, K. You can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki.

⑤ A typical value is **ALERTMANAGER_STORAGE_SIZE=20Gi**. Storage values can be a plain integer or a fixed-point integer using one of these suffixes: E, P, T, G, M, K. You can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki.

- 2 Add values for the retention period, storage class, and storage sizes.

3. Save the file.
4. Apply the changes by running:

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

2.2.5. Additional resources

- [Infrastructure Nodes in OpenShift 4](#)
- [OpenShift Container Platform cluster maximums](#)
- [Creating infrastructure machine sets](#)

2.3. RECOMMENDED ETCD PRACTICES

To ensure optimal performance and scalability for etcd in OpenShift Container Platform, you can complete the following practices.

2.3.1. Storage practices for etcd

Because etcd writes data to disk and persists proposals on disk, its performance depends on disk performance. Although etcd is not particularly I/O intensive, it requires a low latency block device for optimal performance and stability. Because the consensus protocol for etcd depends on persistently storing metadata to a log (WAL), etcd is sensitive to disk-write latency. Slow disks and disk activity from other processes can cause long fsync latencies.

Those latencies can cause etcd to miss heartbeats, not commit new proposals to the disk on time, and ultimately experience request timeouts and temporary leader loss. High write latencies also lead to an OpenShift API slowness, which affects cluster performance. Because of these reasons, avoid colocating other workloads on the control-plane nodes that are I/O sensitive or intensive and share the same underlying I/O infrastructure.

Run etcd on a block device that can write at least 50 IOPS of 8KB sequentially, including fdatasync, in under 10ms. For heavy loaded clusters, sequential 500 IOPS of 8000 bytes (2 ms) are recommended. To measure those numbers, you can use a benchmarking tool, such as the **fio** command.

To achieve such performance, run etcd on machines that are backed by SSD or NVMe disks with low latency and high throughput. Consider single-level cell (SLC) solid-state drives (SSDs), which provide 1 bit per memory cell, are durable and reliable, and are ideal for write-intensive workloads.



NOTE

The load on etcd arises from static factors, such as the number of nodes and pods, and dynamic factors, including changes in endpoints due to pod autoscaling, pod restarts, job executions, and other workload-related events. To accurately size your etcd setup, you must analyze the specific requirements of your workload. Consider the number of nodes, pods, and other relevant factors that impact the load on etcd.

The following hard drive practices provide optimal etcd performance:

- Use dedicated etcd drives. Avoid drives that communicate over the network, such as iSCSI. Do not place log files or other heavy workloads on etcd drives.

- Prefer drives with low latency to support fast read and write operations.
- Prefer high-bandwidth writes for faster compactions and defragmentation.
- Prefer high-bandwidth reads for faster recovery from failures.
- Use solid state drives as a minimum selection. Prefer NVMe drives for production environments.
- Use server-grade hardware for increased reliability.
- Avoid NAS or SAN setups and spinning drives. Ceph Rados Block Device (RBD) and other types of network-attached storage can result in unpredictable network latency. To provide fast storage to etcd nodes at scale, use PCI passthrough to pass NVM devices directly to the nodes.
- Always benchmark by using utilities such as **fio**. You can use such utilities to continuously monitor the cluster performance as it increases.
- Avoid using the Network File System (NFS) protocol or other network based file systems.

Some key metrics to monitor on a deployed OpenShift Container Platform cluster are p99 of etcd disk write ahead log duration and the number of etcd leader changes. Use Prometheus to track these metrics.



NOTE

The etcd member database sizes can vary in a cluster during normal operations. This difference does not affect cluster upgrades, even if the leader size is different from the other members.

2.3.2. Validating the hardware for etcd

To validate the hardware for etcd before or after you create the OpenShift Container Platform cluster, you can use fio.

Prerequisites

- Container runtimes such as Podman or Docker are installed on the machine that you are testing.
- Data is written to the **/var/lib/etcd** path.

Procedure

- Run fio and analyze the results:
 - If you use Podman, run this command:


```
$ sudo podman run --volume /var/lib/etcd:/var/lib/etcd:Z quay.io/cloud-bulldozer/etcd-perf
```
 - If you use Docker, run this command:


```
$ sudo docker run --volume /var/lib/etcd:/var/lib/etcd:Z quay.io/cloud-bulldozer/etcd-perf
```

The output reports whether the disk is fast enough to host etcd by comparing the 99th percentile of the fsync metric captured from the run to see if it is less than 10 ms. A few of the most important etcd metrics that might be affected by I/O performance are as follows:

- **etcd_disk_wal_fsync_duration_seconds_bucket** metric reports the etcd's WAL fsync duration
- **etcd_disk_backend_commit_duration_seconds_bucket** metric reports the etcd backend commit latency duration
- **etcd_server_leader_changes_seen_total** metric reports the leader changes

Because etcd replicates the requests among all the members, its performance strongly depends on network input/output (I/O) latency. High network latencies result in etcd heartbeats taking longer than the election timeout, which results in leader elections that are disruptive to the cluster. A key metric to monitor on a deployed OpenShift Container Platform cluster is the 99th percentile of etcd network peer latency on each etcd cluster member. Use Prometheus to track the metric.

The **histogram_quantile(0.99, rate(etcd_network_peer_round_trip_time_seconds_bucket[2m]))** metric reports the round trip time for etcd to finish replicating the client requests between the members. Ensure that it is less than 50 ms.

Additional resources

- How to use [fio](#) to check etcd disk performance in OpenShift Container Platform
- [etcd performance troubleshooting guide for OpenShift Container Platform](#)

2.3.3. Node scaling for etcd

In general, clusters must have 3 control plane nodes. However, if your cluster is installed on a bare metal platform, it can have up to 5 control plane nodes. If an existing bare-metal cluster has fewer than 5 control plane nodes, you can scale the cluster up as a postinstallation task.

For example, to scale from 3 to 4 control plane nodes after installation, you can add a host and install it as a control plane node. Then, the etcd Operator scales accordingly to account for the additional control plane node.

Scaling a cluster to 4 or 5 control plane nodes is available only on bare metal platforms.

For more information about how to scale control plane nodes by using the Assisted Installer, see "Adding hosts with the API" and "Installing a primary control plane node on a healthy cluster".

The following table shows failure tolerance for clusters of different sizes:

Table 2.2. Failure tolerances by cluster size

Cluster size	Majority	Failure tolerance
1 node	1	0
3 nodes	2	1
4 nodes	3	1
5 nodes	3	2

For more information about recovering from quorum loss, see "Restoring to a previous cluster state".

Additional resources

- [Adding hosts with the API](#)
- [Installing a primary control plane node on a healthy cluster](#)
- [Expanding the cluster](#)
- [Restoring to a previous cluster state](#)

2.3.4. Moving etcd to a different disk

You can move etcd from a shared disk to a separate disk to prevent or resolve performance issues.

The Machine Config Operator (MCO) is responsible for mounting a secondary disk for OpenShift Container Platform 4.18 container storage.



NOTE

This encoded script only supports device names for the following device types:

SCSI or SATA

/dev/sd*

Virtual device

/dev/vd*

NVMe

/dev/nvme*[0-9]*n*

Limitations

- When the new disk is attached to the cluster, the etcd database is part of the root mount. It is not part of the secondary disk or the intended disk when the primary node is recreated. As a result, the primary node will not create a separate **/var/lib/etcd** mount.

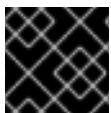
Prerequisites

- You have a backup of your cluster's etcd data.
- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster with **cluster-admin** privileges.
- Add additional disks before uploading the machine configuration.
- The **MachineConfigPool** must match **metadata.labels[machineconfiguration.openshift.io/role]**. This applies to a controller, worker, or a custom pool.



NOTE

This procedure does not move parts of the root file system, such as **/var/**, to another disk or partition on an installed node.



IMPORTANT

This procedure is not supported when using control plane machine sets.

Procedure

1. Attach the new disk to the cluster and verify that the disk is detected in the node by running the **lsblk** command in a debug shell:

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

Note the device name of the new disk reported by the **lsblk** command.

2. Create the following script and name it **etcd-find-secondary-device.sh**:

```
#!/bin/bash
set -uo pipefail

for device in <device_type_glob>; do ①
/usr/sbin/blkid "${device}" &> /dev/null
if [ $? == 2 ]; then
    echo "secondary device found ${device}"
    echo "creating filesystem for etcd mount"
    mkfs.xfs -L var-lib-etcd -f "${device}" &> /dev/null
    udevadm settle
    touch /etc/var-lib-etcd-mount
    exit
fi
done
echo "Couldn't find secondary block device!" >&2
exit 77
```

- 1 Replace **<device_type_glob>** with a shell glob for your block device type. For SCSI or SATA drives, use **/dev/sd***; for virtual drives, use **/dev/vd***; for NVMe drives, use **/dev/nvme*[0-9]*n***.

3. Create a base64-encoded string from the **etcd-find-secondary-device.sh** script and note its contents:

```
$ base64 -w0 etcd-find-secondary-device.sh
```

4. Create a **MachineConfig** YAML file named **etcd-mc.yml** with contents such as the following:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 98-var-lib-etcd
spec:
  config:
```

```

ignition:
  version: 3.4.0
storage:
  files:
    - path: /etc/find-secondary-device
      mode: 0755
      contents:
        source: data:text/plain;charset=utf-8;base64,
<encoded_etcd_find_secondary_device_script> ①

systemd:
  units:
    - name: find-secondary-device.service
      enabled: true
      contents: |
        [Unit]
        Description=Find secondary device
        DefaultDependencies=false
        After=systemd-udev-settle.service
        Before=local-fs-pre.target
        ConditionPathExists=!/etc/var-lib-etcd-mount

        [Service]
        RemainAfterExit=yes
        ExecStart=/etc/find-secondary-device

        RestartForceExitStatus=77

        [Install]
        WantedBy=multi-user.target
    - name: var-lib-etcd.mount
      enabled: true
      contents: |
        [Unit]
        Before=local-fs.target

        [Mount]
        What=/dev/disk/by-label/var-lib-etcd
        Where=/var/lib/etcd
        Type=xfs
        TimeoutSec=120s

        [Install]
        RequiredBy=local-fs.target
    - name: sync-var-lib-etcd-to-etcd.service
      enabled: true
      contents: |
        [Unit]
        Description=Sync etcd data if new mount is empty
        DefaultDependencies=no
        After=var-lib-etcd.mount var.mount
        Before=crio.service

        [Service]
        Type=oneshot
        RemainAfterExit=yes
        ExecCondition=/usr/bin/test ! -d /var/lib/etcd/member

```

```

ExecStart=/usr/sbin/setsebool -P rsync_full_access 1
ExecStart=/bin/rsync -ar /sysroot/ostree/deploy/rhcos/var/lib/etcd/ /var/lib/etcd/
ExecStart=/usr/sbin/semanage fcontext -a -t container_var_lib_t '/var/lib/etcd(.*)?'
ExecStart=/usr/sbin/setsebool -P rsync_full_access 0
TimeoutSec=0

[Install]
WantedBy=multi-user.target graphical.target
- name: restorecon-var-lib-etcd.service
  enabled: true
  contents: |
    [Unit]
    Description=Restore recursive SELinux security contexts
    DefaultDependencies=no
    After=var-lib-etcd.mount
    Before=crio.service

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/sbin/restorecon -R /var/lib/etcd/
TimeoutSec=0

[Install]
WantedBy=multi-user.target graphical.target

```

- 1 Replace <encoded_etcd_find_secondary_device_script> with the encoded script contents that you noted.

5. Apply the created **MachineConfig** YAML file:

```
$ oc create -f etcd-mc.yml
```

Verification steps

- Run the **grep /var/lib/etcd /proc/mounts** command in a debug shell for the node to ensure that the disk is mounted:

```
$ oc debug node/<node_name>
# grep -w "/var/lib/etcd" /proc/mounts
```

Example output

```
/dev/sdb /var/lib/etcd xfs rw,seclabel,relatime,attr2,inode64,logbufs=8,logbsize=32k,noquota
0 0
```

Additional resources

- [Red Hat Enterprise Linux CoreOS \(RHCOS\)](#)

2.3.5. Defragmenting etcd data

For large and dense clusters, etcd can suffer from poor performance if the keyspace grows too large and exceeds the space quota. Periodically maintain and defragment etcd to free up space in the data store. Monitor Prometheus for etcd metrics and defragment it when required; otherwise, etcd can raise a cluster-wide alarm that puts the cluster into a maintenance mode that accepts only key reads and deletes.

Monitor these key metrics:

- **etcd_server_quota_backend_bytes**, which is the current quota limit
- **etcd_mvcc_db_total_size_in_use_in_bytes**, which indicates the actual database usage after a history compaction
- **etcd_mvcc_db_total_size_in_bytes**, which shows the database size, including free space waiting for defragmentation

Defragment etcd data to reclaim disk space after events that cause disk fragmentation, such as etcd history compaction.

History compaction is performed automatically every five minutes and leaves gaps in the back-end database. This fragmented space is available for use by etcd, but is not available to the host file system. You must defragment etcd to make this space available to the host file system.

Defragmentation occurs automatically, but you can also trigger it manually.



NOTE

Automatic defragmentation is good for most cases, because the etcd operator uses cluster information to determine the most efficient operation for the user.

2.3.5.1. Automatic defragmentation

The etcd Operator automatically defragments disks. No manual intervention is needed.

Verify that the defragmentation process is successful by viewing one of these logs:

- etcd logs
- cluster-etcd-operator pod
- operator status error log



WARNING

Automatic defragmentation can cause leader election failure in various OpenShift core components, such as the Kubernetes controller manager, which triggers a restart of the failing component. The restart is harmless and either triggers failover to the next running instance or the component resumes work again after the restart.

Example log output for successful defragmentation

etcd member has been defragmented: <member_name>, memberID: <member_id>

Example log output for unsuccessful defragmentation

failed defrag on member: <member_name>, memberID: <member_id>: <error_message>

2.3.5.2. Manual defragmentation

A Prometheus alert indicates when you need to use manual defragmentation. The alert is displayed in two cases:

- When etcd uses more than 50% of its available space for more than 10 minutes
- When etcd is actively using less than 50% of its total database size for more than 10 minutes

You can also determine whether defragmentation is needed by checking the etcd database size in MB that will be freed by defragmentation with the PromQL expression:

(etcd_mvcc_db_total_size_in_bytes - etcd_mvcc_db_total_size_in_use_in_bytes)/1024/1024



WARNING

Defragmenting etcd is a blocking action. The etcd member will not respond until defragmentation is complete. For this reason, wait at least one minute between defragmentation actions on each of the pods to allow the cluster to recover.

Follow this procedure to defragment etcd data on each etcd member.

Prerequisites

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

Procedure

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

```
$ oc -n openshift-etcd get pods -l k8s-app=etcd -o wide
```

Example output

etcd-ip-10-0-159-225.example.redhat.com	3/3	Running	0	175m
10.0.159.225	ip-10-0-159-225.example.redhat.com	<none>	<none>	
etcd-ip-10-0-191-37.example.redhat.com	3/3	Running	0	173m
10.0.191.37	ip-10-0-191-37.example.redhat.com	<none>	<none>	
etcd-ip-10-0-199-170.example.redhat.com	3/3	Running	0	176m
10.0.199.170	ip-10-0-199-170.example.redhat.com	<none>	<none>	

- b. Choose a pod and run the following command to determine which etcd member is the leader:

```
$ oc rsh -n openshift-etcd etcd-ip-10-0-159-225.example.redhat.com etcdctl endpoint status --cluster -w table
```

Example output

```
Defaulting container name to etcdctl.
Use 'oc describe pod/etcd-ip-10-0-159-225.example.redhat.com -n openshift-etcd' to see all of the containers in this pod.

+-----+-----+-----+-----+-----+
+-----+-----+
| ENDPOINT | ID | VERSION | DB SIZE | IS LEADER | IS LEARNER |
| RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+-----+-----+-----+-----+
+-----+-----+
| https://10.0.191.37:2379 | 251cd44483d811c3 | 3.5.9 | 104 MB | false | false |
7 | 91624 | 91624 | |
| https://10.0.159.225:2379 | 264c7c58ecbdabee | 3.5.9 | 104 MB | false | false |
7 | 91624 | 91624 | |
| https://10.0.199.170:2379 | 9ac311f93915cc79 | 3.5.9 | 104 MB | true | false |
7 | 91624 | 91624 | |
+-----+-----+-----+-----+
+-----+-----+
```

Based on the **IS LEADER** column of this output, the **https://10.0.199.170:2379** endpoint is the leader. Matching this endpoint with the output of the previous step, the pod name of the leader is **etcd-ip-10-0-199-170.example.redhat.com**.

2. Defragment an etcd member.

- a. Connect to the running etcd container, passing in the name of a pod that is *not* the leader:

```
$ oc rsh -n openshift-etcd etcd-ip-10-0-159-225.example.redhat.com
```

- b. Unset the **ETCDCTL_ENDPOINTS** environment variable:

```
sh-4.4# unset ETCDCTL_ENDPOINTS
```

- c. Defragment the etcd member:

```
sh-4.4# etcdctl --command-timeout=30s --endpoints=https://localhost:2379 defrag
```

Example output

```
Finished defragmenting etcd member[https://localhost:2379]
```

If a timeout error occurs, increase the value for **--command-timeout** until the command succeeds.

- d. Verify that the database size was reduced:

```
sh-4.4# etcdctl endpoint status -w table --cluster
```

Example output

ENDPOINT	ID	VERSION	DB SIZE	IS LEADER	IS LEARNER	RAFT TERM	RAFT INDEX	RAFT APPLIED INDEX	ERRORS
https://10.0.191.37:2379	251cd44483d811c3	3.5.9	104 MB	false	false	7	91624	91624	
https://10.0.159.225:2379	264c7c58ecbdabee	3.5.9	41 MB	false	false	7	91624	91624	1
https://10.0.199.170:2379	9ac311f93915cc79	3.5.9	104 MB	true	false	7	91624	91624	

This example shows that the database size for this etcd member is now 41 MB as opposed to the starting size of 104 MB.

- e. Repeat these steps to connect to each of the other etcd members and defragment them. Always defragment the leader last.
Wait at least one minute between defragmentation actions to allow the etcd pod to recover. Until the etcd pod recovers, the etcd member will not respond.
- 3. If any **NOSPACE** alarms were triggered due to the space quota being exceeded, clear them.
 - a. Check if there are any **NOSPACE** alarms:

```
sh-4.4# etcdctl alarm list
```

Example output

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

- b. Clear the alarms:

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

2.3.6. Setting tuning parameters for etcd

You can set the control plane hardware speed to "**Standard**", "**Slower**", or the default, which is "".

The default setting allows the system to decide which speed to use. This value enables upgrades from versions where this feature does not exist, as the system can select values from previous versions.

By selecting one of the other values, you are overriding the default. If you see many leader elections due to timeouts or missed heartbeats and your system is set to "" or "**Standard**", set the hardware speed to "**Slower**" to make the system more tolerant to the increased latency.

2.3.6.1. Changing hardware speed tolerance

To change the hardware speed tolerance for etcd, complete the following steps.

Procedure

- Check to see what the current value is by entering the following command:

```
$ oc describe etcd/cluster | grep "Control Plane Hardware Speed"
```

Example output

```
Control Plane Hardware Speed: <VALUE>
```



NOTE

If the output is empty, the field has not been set and should be considered as the default ("").

- Change the value by entering the following command. Replace **<value>** with one of the valid values: "", "**Standard**", or "**Slower**":

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"controlPlaneHardwareSpeed": "<value>"} }'
```

The following table indicates the heartbeat interval and leader election timeout for each profile. These values are subject to change.

Profile	ETCD_HEARTBEAT_INTERVAL	ETCD_LEADER_ELECTION_TIMEOUT
""	Varies depending on platform	Varies depending on platform
Standard	100	1000
Slower	500	2500

- Review the output:

Example output

```
etcd.operator.openshift.io/cluster patched
```

If you enter any value besides the valid values, error output is displayed. For example, if you entered "**Faster**" as the value, the output is as follows:

Example output

```
The Etcd "cluster" is invalid: spec.controlPlaneHardwareSpeed: Unsupported value: "Faster": supported values: "", "Standard", "Slower"
```

- Verify that the value was changed by entering the following command:

```
$ oc describe etcd/cluster | grep "Control Plane Hardware Speed"
```

Example output

```
Control Plane Hardware Speed: ""
```

5. Wait for etcd pods to roll out:

```
$ oc get pods -n openshift-etcd -w
```

The following output shows the expected entries for master-0. Before you continue, wait until all masters show a status of **4/4 Running**.

Example output

installer-9-ci-ln-qkgs94t-72292-9clnd-master-0	0/1	Pending	0	0s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0	0/1	Pending	0	0s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0	0/1	ContainerCreating	0	0s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0	0/1	ContainerCreating	0	1s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0	1/1	Running	0	2s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0	0/1	Completed	0	34s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0	0/1	Completed	0	36s
installer-9-ci-ln-qkgs94t-72292-9clnd-master-0	0/1	Completed	0	36s
etcd-guard-ci-ln-qkgs94t-72292-9clnd-master-0	0/1	Running	0	26m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0	4/4	Terminating	0	11m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0	4/4	Terminating	0	11m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0	0/4	Pending	0	0s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0	0/4	Init:1/3	0	1s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0	0/4	Init:2/3	0	2s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0	0/4	PodInitializing	0	3s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0	3/4	Running	0	4s
etcd-guard-ci-ln-qkgs94t-72292-9clnd-master-0	1/1	Running	0	26m
etcd-ci-ln-qkgs94t-72292-9clnd-master-0	3/4	Running	0	20s
etcd-ci-ln-qkgs94t-72292-9clnd-master-0	4/4	Running	0	20s

6. Enter the following command to review the values:

```
$ oc describe -n openshift-etcd pod/<ETCD_PODNAME> | grep -e HEARTBEAT_INTERVAL  
-e ELECTION_TIMEOUT
```



NOTE

These values might not have changed from the default.

Additional resources

- [Understanding feature gates](#)

2.3.7. Increasing the database size for etcd

You can set the disk quota in gibibytes (GiB) for each etcd instance. If you set a disk quota for your etcd instance, you can specify integer values from 8 to 32. The default value is 8. You can specify only increasing values.

You might want to increase the disk quota if you encounter a **low space** alert. This alert indicates that the cluster is too large to fit in etcd despite automatic compaction and defragmentation. If you see this alert, you need to increase the disk quota immediately because after etcd runs out of space, writes fail.

Another scenario where you might want to increase the disk quota is if you encounter an **excessive database growth** alert. This alert is a warning that the database might grow too large in the next four hours. In this scenario, consider increasing the disk quota so that you do not eventually encounter a **low space** alert and possible write fails.

If you increase the disk quota, the disk space that you specify is not immediately reserved. Instead, etcd can grow to that size if needed. Ensure that etcd is running on a dedicated disk that is larger than the value that you specify for the disk quota.

For large etcd databases, the control plane nodes must have additional memory and storage. Because you must account for the API server cache, the minimum memory required is at least three times the configured size of the etcd database.



IMPORTANT

Increasing the database size for etcd 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](#).

2.3.7.1. Changing the etcd database size

To change the database size for etcd, complete the following steps.

Procedure

- Check the current value of the disk quota for each etcd instance by entering the following command:

```
$ oc describe etcd/cluster | grep "Backend Quota"
```

Example output

```
Backend Quota Gi B: <value>
```

- Change the value of the disk quota by entering the following command:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"backendQuotaGiB": <value>}}'
```

Example output

```
etcd.operator.openshift.io/cluster patched
```

Verification

- Verify that the new value for the disk quota is set by entering the following command:

```
$ oc describe etcd/cluster | grep "Backend Quota"
```

The etcd Operator automatically rolls out the etcd instances with the new values.

- Verify that the etcd pods are up and running by entering the following command:

```
$ oc get pods -n openshift-etcd
```

The following output shows the expected entries.

Example output

NAME	READY	STATUS	RESTARTS	AGE
etcd-ci-ln-b6kfs2-72292-mzwbq-master-0	4/4	Running	0	39m
etcd-ci-ln-b6kfs2-72292-mzwbq-master-1	4/4	Running	0	37m
etcd-ci-ln-b6kfs2-72292-mzwbq-master-2	4/4	Running	0	41m
etcd-guard-ci-ln-b6kfs2-72292-mzwbq-master-0	1/1	Running	0	51m
etcd-guard-ci-ln-b6kfs2-72292-mzwbq-master-1	1/1	Running	0	49m
etcd-guard-ci-ln-b6kfs2-72292-mzwbq-master-2	1/1	Running	0	54m
installer-5-ci-ln-b6kfs2-72292-mzwbq-master-1	0/1	Completed	0	51m
installer-7-ci-ln-b6kfs2-72292-mzwbq-master-0	0/1	Completed	0	46m
installer-7-ci-ln-b6kfs2-72292-mzwbq-master-1	0/1	Completed	0	44m
installer-7-ci-ln-b6kfs2-72292-mzwbq-master-2	0/1	Completed	0	49m
installer-8-ci-ln-b6kfs2-72292-mzwbq-master-0	0/1	Completed	0	40m
installer-8-ci-ln-b6kfs2-72292-mzwbq-master-1	0/1	Completed	0	38m
installer-8-ci-ln-b6kfs2-72292-mzwbq-master-2	0/1	Completed	0	42m
revision-pruner-7-ci-ln-b6kfs2-72292-mzwbq-master-0	0/1	Completed	0	43m
revision-pruner-7-ci-ln-b6kfs2-72292-mzwbq-master-1	0/1	Completed	0	43m
revision-pruner-7-ci-ln-b6kfs2-72292-mzwbq-master-2	0/1	Completed	0	43m
revision-pruner-8-ci-ln-b6kfs2-72292-mzwbq-master-0	0/1	Completed	0	42m
revision-pruner-8-ci-ln-b6kfs2-72292-mzwbq-master-1	0/1	Completed	0	42m
revision-pruner-8-ci-ln-b6kfs2-72292-mzwbq-master-2	0/1	Completed	0	42m

- Verify that the disk quota value is updated for the etcd pod by entering the following command:

```
$ oc describe -n openshift-etcd pod/<etcd_podname> | grep "ETCD_QUOTA_BACKEND_BYTES"
```

The value might not have changed from the default value of **8**.

Example output

```
ETCD_QUOTA_BACKEND_BYTES: 8589934592
```



NOTE

While the value that you set is an integer in GiB, the value shown in the output is converted to bytes.

2.3.7.2. Troubleshooting

If you encounter issues when you try to increase the database size for etcd, the following troubleshooting steps might help.

2.3.7.2.1. Value is too small

If the value that you specify is less than **8**, you see the following error message:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"backendQuotaGiB": 5}}'
```

Example error message

The Etcd "cluster" is invalid:

- * spec.backendQuotaGiB: Invalid value: 5: spec.backendQuotaGiB in body should be greater than or equal to 8
- * spec.backendQuotaGiB: Invalid value: "integer": etcd backendQuotaGiB may not be decreased

To resolve this issue, specify an integer between **8** and **32**.

2.3.7.2.2. Value is too large

If the value that you specify is greater than **32**, you see the following error message:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"backendQuotaGiB": 64}}'
```

Example error message

The Etcd "cluster" is invalid: spec.backendQuotaGiB: Invalid value: 64: spec.backendQuotaGiB in body should be less than or equal to 32

To resolve this issue, specify an integer between **8** and **32**.

2.3.7.2.3. Value is decreasing

If the value is set to a valid value between **8** and **32**, you cannot decrease the value. Otherwise, you see an error message.

1. Check to see the current value by entering the following command:

```
$ oc describe etcd/cluster | grep "Backend Quota"
```

Example output

```
Backend Quota Gi B: 10
```

2. Decrease the disk quota value by entering the following command:

```
$ oc patch etcd/cluster --type=merge -p '{"spec": {"backendQuotaGiB": 8}}'
```

Example error message

The Etcd "cluster" is invalid: spec.backendQuotaGiB: Invalid value: "integer": etcd backendQuotaGiB may not be decreased

3. To resolve this issue, specify an integer greater than **10**.

CHAPTER 3. TELCO CORE REFERENCE DESIGN SPECIFICATIONS

The telco core reference design specifications (RDS) configures an OpenShift Container Platform cluster running on commodity hardware to host telco core workloads.

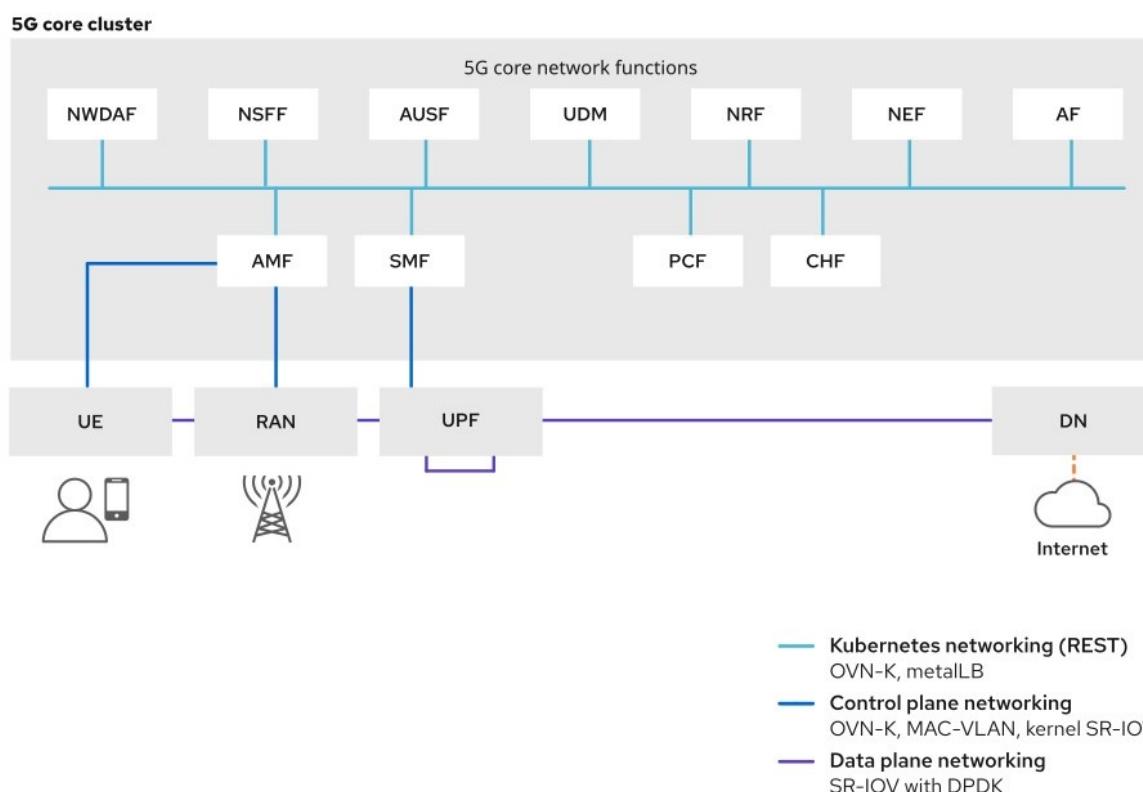
3.1. TELCO CORE RDS 4.18 USE MODEL OVERVIEW

The telco core reference design specifications (RDS) describes a platform that supports large-scale telco applications, including control plane functions such as signaling and aggregation. It also includes some centralized data plane functions, such as user plane functions (UPF). These functions generally require scalability, complex networking support, resilient software-defined storage, and support performance requirements that are less stringent and constrained than far-edge deployments such as RAN.

3.2. ABOUT THE TELCO CORE CLUSTER USE MODEL

The telco core cluster use model is designed for clusters that run on commodity hardware. Telco core clusters support large scale telco applications including control plane functions such as signaling, aggregation, and session border controller (SBC); and centralized data plane functions such as 5G user plane functions (UPF). Telco core cluster functions require scalability, complex networking support, resilient software-defined storage, and support performance requirements that are less stringent and constrained than far-edge RAN deployments.

Figure 3.1. Telco core RDS cluster service-based architecture and networking topology



Networking requirements for telco core functions vary widely across a range of networking features and performance points. IPv6 is a requirement and dual-stack is common. Some functions need maximum throughput and transaction rate and require support for user-plane DPDK networking. Other functions

use more typical cloud-native patterns and can rely on OVN-Kubernetes, kernel networking, and load balancing.

Telco core clusters are configured as standard with three control plane and two or more worker nodes configured with the stock (non-RT) kernel. In support of workloads with varying networking and performance requirements, you can segment worker nodes by using **MachineConfigPool** custom resources (CR), for example, for non-user data plane or high-throughput use cases. In support of required telco operational features, core clusters have a standard set of Day 2 OLM-managed Operators installed.

3.3. REFERENCE DESIGN SCOPE

The telco core, telco RAN and telco hub reference design specifications (RDS) capture the recommended, tested, and supported configurations to get reliable and repeatable performance for clusters running the telco core and telco RAN profiles.

Each RDS includes the released features and supported configurations that are engineered and validated for clusters to run the individual profiles. The configurations provide a baseline OpenShift Container Platform installation that meets feature and KPI targets. Each RDS also describes expected variations for each individual configuration. Validation of each RDS includes many long duration and at-scale tests.



NOTE

The validated reference configurations are updated for each major Y-stream release of OpenShift Container Platform. Z-stream patch releases are periodically re-tested against the reference configurations.

3.4. DEVIATIONS FROM THE REFERENCE DESIGN

Deviating from the validated telco core, telco RAN DU, and telco hub reference design specifications (RDS) can have significant impact beyond the specific component or feature that you change. Deviations require analysis and engineering in the context of the complete solution.



IMPORTANT

All deviations from the RDS should be analyzed and documented with clear action tracking information. Due diligence is expected from partners to understand how to bring deviations into line with the reference design. This might require partners to provide additional resources to engage with Red Hat to work towards enabling their use case to achieve a best in class outcome with the platform. This is critical for the supportability of the solution and ensuring alignment across Red Hat and with partners.

Deviation from the RDS can have some or all of the following consequences:

- It can take longer to resolve issues.
- There is a risk of missing project service-level agreements (SLAs), project deadlines, end provider performance requirements, and so on.
- Unapproved deviations may require escalation at executive levels.

**NOTE**

Red Hat prioritizes the servicing of requests for deviations based on partner engagement priorities.

3.5. TELCO CORE COMMON BASELINE MODEL

The following configurations and use models are applicable to all telco core use cases. The telco core use cases build on this common baseline of features.

Cluster topology

Telco core clusters conform to the following requirements:

- High availability control plane (three or more control plane nodes)
- Non-schedulable control plane nodes
- Multiple machine config pools

Storage

Telco core use cases require persistent storage as provided by Red Hat OpenShift Data Foundation.

Networking

Telco core cluster networking conforms to the following requirements:

- Dual stack IPv4/IPv6 (IPv4 primary).
- Fully disconnected – clusters do not have access to public networking at any point in their lifecycle.
- Supports multiple networks. Segmented networking provides isolation between operations, administration and maintenance (OAM), signaling, and storage traffic.
- Cluster network type is OVN-Kubernetes as required for IPv6 support.
- Telco core clusters have multiple layers of networking supported by underlying RHCOS, SR-IOV Network Operator, Load Balancer and other components. These layers include the following:
 - Cluster networking layer. The cluster network configuration is defined and applied through the installation configuration. Update the configuration during Day 2 operations with the NMState Operator. Use the initial configuration to establish the following:
 - Host interface configuration.
 - Active/active bonding (LACP).
 - Secondary/additional network layer. Configure the OpenShift Container Platform CNI through network **additionalNetwork** or **NetworkAttachmentDefinition** CRs. Use the initial configuration to configure MACVLAN virtual network interfaces.
 - Application workload layer. User plane networking runs in cloud-native network functions (CNFs).

Service Mesh

Telco CNFs can use Service Mesh. All telco core clusters require a Service Mesh implementation. The choice of implementation and configuration is outside the scope of this specification.

3.6. TELCO CORE CLUSTER COMMON USE MODEL ENGINEERING CONSIDERATIONS

- Cluster workloads are detailed in "Application workloads".
- Worker nodes should run on either of the following CPUs:
 - Intel 3rd Generation Xeon (IceLake) CPUs or better when supported by OpenShift Container Platform, or CPUs with the silicon security bug (Spectre and similar) mitigations turned off. Skylake and older CPUs can experience 40% transaction performance drops when Spectre and similar mitigations are enabled.
 - AMD EPYC Zen 4 CPUs (Genoa, Bergamo, or newer) or better when supported by OpenShift Container Platform.



NOTE

Currently, per-pod power management is not available for AMD CPUs.

- IRQ balancing is enabled on worker nodes. The **PerformanceProfile** CR sets **globallyDisableIrqLoadBalancing** to false. Guaranteed QoS pods are annotated to ensure isolation as described in "CPU partitioning and performance tuning".
- All cluster nodes should have the following features:
 - Have Hyper-Threading enabled
 - Have x86_64 CPU architecture
 - Have the stock (non-realtime) kernel enabled
 - Are not configured for workload partitioning
- The balance between power management and maximum performance varies between machine config pools in the cluster. The following configurations should be consistent for all nodes in a machine config pools group.
 - Cluster scaling. See "Scalability" for more information.
 - Clusters should be able to scale to at least 120 nodes.
- CPU partitioning is configured using a **PerformanceProfile** CR and is applied to nodes on a per **MachineConfigPool** basis. See "CPU partitioning and performance tuning" for additional considerations.
- CPU requirements for OpenShift Container Platform depend on the configured feature set and application workload characteristics. For a cluster configured according to the reference configuration running a simulated workload of 3000 pods as created by the kube-burner node-density test, the following CPU requirements are validated:
 - The minimum number of reserved CPUs for control plane and worker nodes is 2 CPUs (4 hyper-threads) per NUMA node.

- The NICs used for non-DPDK network traffic should be configured to use at least 16 RX/TX queues.
- Nodes with large numbers of pods or other resources might require additional reserved CPUs. The remaining CPUs are available for user workloads.



NOTE

Variations in OpenShift Container Platform configuration, workload size, and workload characteristics require additional analysis to determine the effect on the number of required CPUs for the OpenShift platform.

3.6.1. Application workloads

Application workloads running on telco core clusters can include a mix of high performance cloud-native network functions (CNFs) and traditional best-effort or burstable pod workloads.

Guaranteed QoS scheduling is available to pods that require exclusive or dedicated use of CPUs due to performance or security requirements. Typically, pods that run high performance or latency sensitive CNFs by using user plane networking (for example, DPDK) require exclusive use of dedicated whole CPUs achieved through node tuning and guaranteed QoS scheduling. When creating pod configurations that require exclusive CPUs, be aware of the potential implications of hyper-threaded systems. Pods should request multiples of 2 CPUs when the entire core (2 hyper-threads) must be allocated to the pod.

Pods running network functions that do not require high throughput or low latency networking should be scheduled with best-effort or burstable QoS pods and do not require dedicated or isolated CPU cores.

Engineering considerations

Use the following information to plan telco core workloads and cluster resources:

- CNF applications should conform to the latest version of Red Hat Best Practices for [Kubernetes](#).
- Use a mix of best-effort and burstable QoS pods as required by your applications.
 - Use guaranteed QoS pods with proper configuration of reserved or isolated CPUs in the **PerformanceProfile** CR that configures the node.
 - Guaranteed QoS Pods must include annotations for fully isolating CPUs.
 - Best effort and burstable pods are not guaranteed exclusive CPU use. Workloads can be preempted by other workloads, operating system daemons, or kernel tasks.
- Use exec probes sparingly and only when no other suitable option is available.
 - Do not use exec probes if a CNF uses CPU pinning. Use other probe implementations, for example, **httpGet** or **tcpSocket**.
 - When you need to use exec probes, limit the exec probe frequency and quantity. The maximum number of exec probes must be kept below 10, and the frequency must not be set to less than 10 seconds.
 - You can use startup probes, because they do not use significant resources at steady-state operation. This limitation on exec probes applies primarily to liveness and readiness probes. Exec probes cause much higher CPU usage on management cores compared to other probe types because they require process forking.

3.6.2. Signaling workloads

Signaling workloads typically use SCTP, REST, gRPC or similar TCP or UDP protocols. Signaling workloads support hundreds of thousands of transactions per second (TPS) by using a secondary multus CNI configured as MACVLAN or SR-IOV interface. These workloads can run in pods with either guaranteed or burstable QoS.

3.7. TELCO CORE RDS COMPONENTS

The following sections describe the various OpenShift Container Platform components and configurations that you use to configure and deploy clusters to run telco core workloads.

3.7.1. CPU partitioning and performance tuning

New in this release

- No reference design updates in this release

Description

CPU partitioning improves performance and reduces latency by separating sensitive workloads from general-purpose tasks, interrupts, and driver work queues. The CPUs allocated to those auxiliary processes are referred to as *reserved* in the following sections. In a system with Hyper-Threading enabled, a CPU is one hyper-thread.

Limits and requirements

- The operating system needs a certain amount of CPU to perform all the support tasks, including kernel networking.
 - A system with just user plane networking applications (DPDK) needs at least one core (2 hyper-threads when enabled) reserved for the operating system and the infrastructure components.
- In a system with Hyper-Threading enabled, core sibling threads must always be in the same pool of CPUs.
- The set of reserved and isolated cores must include all CPU cores.
- Core 0 of each NUMA node must be included in the reserved CPU set.
- Low latency workloads require special configuration to avoid being affected by interrupts, kernel scheduler, or other parts of the platform. For more information, see "Creating a performance profile".

Engineering considerations

- The minimum reserved capacity (**systemReserved**) required can be found by following the guidance in the [Which amount of CPU and memory are recommended to reserve for the system in OpenShift 4 nodes?](#) Knowledgebase article.
- The actual required reserved CPU capacity depends on the cluster configuration and workload attributes.
- The reserved CPU value must be rounded up to a full core (2 hyper-threads) alignment.

- Changes to CPU partitioning cause the nodes contained in the relevant machine config pool to be drained and rebooted.
- The reserved CPUs reduce the pod density, because the reserved CPUs are removed from the allocatable capacity of the OpenShift Container Platform node.
- The real-time workload hint should be enabled for real-time capable workloads.
 - Applying the real time **workloadHint** setting results in the **nohz_full** kernel command line parameter being applied to improve performance of high performance applications. When you apply the **workloadHint** setting, any isolated or burstable pods that do not have the **cpu-quota.crio.io: "disable"** annotation and a proper **runtimeClassName** value, are subject to CRI-O rate limiting. When you set the **workloadHint** parameter, be aware of the tradeoff between increased performance and the potential impact of CRI-O rate limiting. Ensure that required pods are correctly annotated.
- Hardware without IRQ affinity support affects isolated CPUs. All server hardware must support IRQ affinity to ensure that pods with guaranteed CPU QoS can fully use allocated CPUs.
- OVS dynamically manages its **cputset** entry to adapt to network traffic needs. You do not need to reserve an additional CPU for handling high network throughput on the primary CNI.
- If workloads running on the cluster use kernel level networking, the RX/TX queue count for the participating NICs should be set to 16 or 32 queues if the hardware permits it. Be aware of the default queue count. With no configuration, the default queue count is one RX/TX queue per online CPU; which can result in too many interrupts being allocated.



NOTE

Some drivers do not deallocate the interrupts even after reducing the queue count.

- If workloads running on the cluster require cgroup v1, you can configure nodes to use cgroup v1 as part of the initial cluster deployment. See "Enabling Linux control group version 1 (cgroup v1)" and [Red Hat Enterprise Linux 9 changes in the context of Red Hat OpenShift workloads](#).



NOTE

Support for cgroup v1 is planned for removal in OpenShift Container Platform 4.19. Clusters running cgroup v1 must transition to cgroup v2.

Additional resources

- [Creating a performance profile](#)
- [Configuring host firmware for low latency and high performance](#)
- [Enabling Linux cgroup v1 during installation](#)

3.7.2. Service mesh

Description

Telco core cloud-native functions (CNFs) typically require a service mesh implementation. Specific service mesh features and performance requirements are dependent on the application. The selection of service mesh implementation and configuration is outside the scope of this documentation. You must account for the impact of service mesh on cluster resource usage and performance, including additional latency introduced in pod networking, in your implementation.

Additional resources

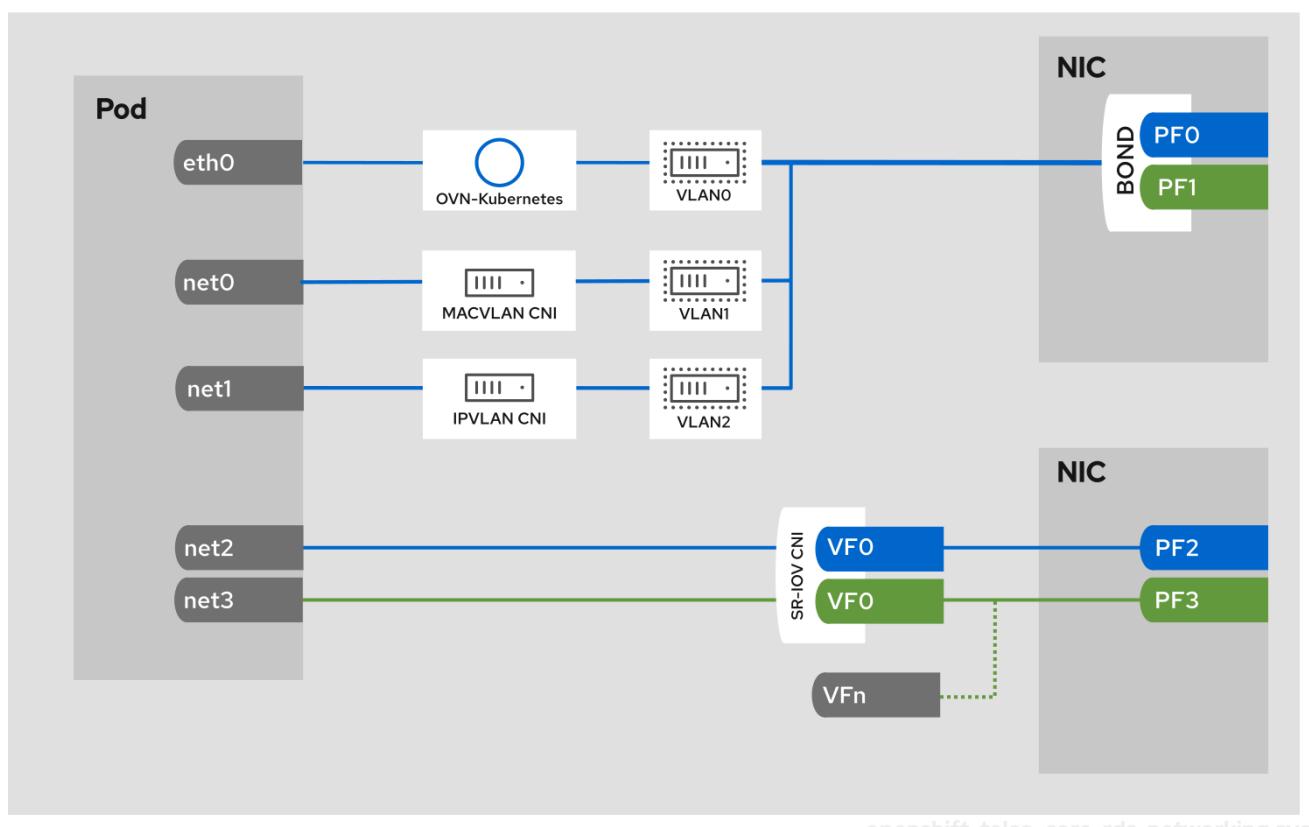
- [About OpenShift Service Mesh](#)

3.7.3. Networking

The following diagram describes the telco core reference design networking configuration.

Figure 3.2. Telco core reference design networking configuration

Node



`openshift-telco-core-rds-networking.svg`

New in this release

- Support for disabling vendor plugins in the SR-IOV Operator
- [New knowledge base article on creating custom node firewall rules](#)
- Extended telco core RDS validation with MetalLB and EgressIP telco QE validation
- FRR-K8s is now available under the Cluster Network Operator.



NOTE

If you have custom **FRRConfiguration** CRs in the **metallb-system** namespace, you must move them under the **openshift-network-operator** namespace.

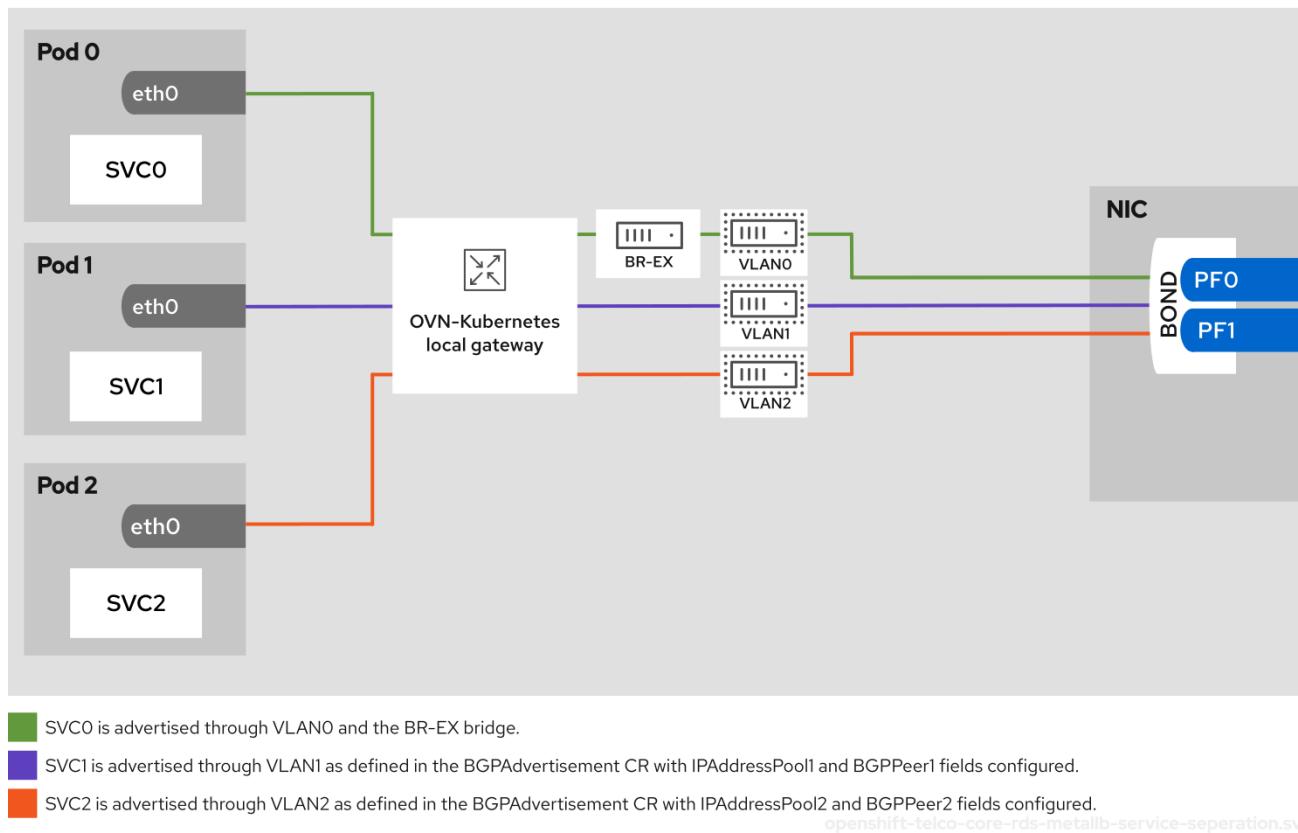
Description

- The cluster is configured for dual-stack IP (IPv4 and IPv6).
- The validated physical network configuration consists of two dual-port NICs. One NIC is shared among the primary CNI (OVN-Kubernetes) and IPVLAN and MACVLAN traffic, while the second one is dedicated to SR-IOV VF-based pod traffic.
- A Linux bonding interface (**bond0**) is created in active-active IEEE 802.3ad LACP mode with the two NIC ports attached. The top-of-rack networking equipment must support and be configured for multi-chassis link aggregation (mLAG) technology.
- VLAN interfaces are created on top of **bond0**, including for the primary CNI.
- Bond and VLAN interfaces are created at cluster install time during the network configuration stage of the installation. Except for the **vlan0** VLAN used by the primary CNI, all other VLANs can be created during Day 2 activities with the Kubernetes NMstate Operator.
- MACVLAN and IPVLAN interfaces are created with their corresponding CNIs. They do not share the same base interface. For more information, see "Cluster Network Operator".
- SR-IOV VFs are managed by the SR-IOV Network Operator.
- To ensure consistent source IP addresses for pods behind a LoadBalancer Service, configure an **EgressIP** CR and specify the **podSelector** parameter.
- You can implement service traffic separation by doing the following:
 - a. Configure VLAN interfaces and specific kernel IP routes on the nodes using **NodeNetworkConfigurationPolicy** CRs.
 - b. Create a MetalLB **BGPPeer** CR for each VLAN to establish peering with the remote BGP router.
 - c. Define a MetalLB **BGPAdvertisement** CR to specify which IP address pools should be advertised to a selected list of **BGPPeer** resources.

The following diagram illustrates how specific service IP addresses are advertised to the outside via specific VLAN interfaces. Services routes are defined in **BGPAdvertisement** CRs and configured with values for **IPAddressPool1** and **BGPPeer1** fields.

Figure 3.3. Telco core reference design MetalLB service separation

Node



Additional resources

- Understanding networking

3.7.3.1. Cluster Network Operator

New in this release

- No reference design updates in this release

Description

The Cluster Network Operator (CNO) deploys and manages the cluster network components including the default OVN-Kubernetes network plugin during cluster installation. The CNO allows for configuring primary interface MTU settings, OVN gateway modes to use node routing tables for pod egress, and additional secondary networks such as MACVLAN.

In support of network traffic separation, multiple network interfaces are configured through the CNO. Traffic steering to these interfaces is configured through static routes applied by using the NMState Operator. To ensure that pod traffic is properly routed, OVN-K is configured with the **routingViaHost** option enabled. This setting uses the kernel routing table and the applied static routes rather than OVN for pod egress traffic.

The Whereabouts CNI plugin is used to provide dynamic IPv4 and IPv6 addressing for additional pod network interfaces without the use of a DHCP server.

Limits and requirements

- OVN-Kubernetes is required for IPv6 support.

- Large MTU cluster support requires connected network equipment to be set to the same or larger value. MTU size up to 8900 is supported.
- MACVLAN and IPVLAN cannot co-locate on the same main interface due to their reliance on the same underlying kernel mechanism, specifically the **rx_handler**. This handler allows a third-party module to process incoming packets before the host processes them, and only one such handler can be registered per network interface. Since both MACVLAN and IPVLAN need to register their own **rx_handler** to function, they conflict and cannot coexist on the same interface. Review the source code for more details:
 - [linux/v6.10.2/source/drivers/net/ipvlan/ipvlan_main.c#L82](https://github.com/01org/01-kernel/blob/v6.10.2/source/drivers/net/ipvlan/ipvlan_main.c#L82)
 - [linux/v6.10.2/source/drivers/net/macvlan.c#L1260](https://github.com/01org/01-kernel/blob/v6.10.2/source/drivers/net/macvlan.c#L1260)
- Alternative NIC configurations include splitting the shared NIC into multiple NICs or using a single dual-port NIC, though they have not been tested and validated.
- Clusters with single-stack IP configuration are not validated.
- The **reachabilityTotalTimeoutSeconds** parameter in the **Network** CR configures the **EgressIP** node reachability check total timeout in seconds. The recommended value is **1** second.

Engineering considerations

- Pod egress traffic is handled by kernel routing table using the **routingViaHost** option. Appropriate static routes must be configured in the host.

Additional resources

- [Cluster Network Operator](#)

3.7.3.2. Load balancer

New in this release

- FRR-K8s is now available under the Cluster Network Operator.



IMPORTANT

If you have custom **FRRConfiguration** CRs in the **metallb-system** namespace, you must move them under the **openshift-network-operator** namespace.

Description

MetallLB is a load-balancer implementation for bare metal Kubernetes clusters that uses standard routing protocols. It enables a Kubernetes service to get an external IP address which is also added to the host network for the cluster. The MetalLB Operator deploys and manages the lifecycle of a MetalLB instance in a cluster. Some use cases might require features not available in MetalLB, such as stateful load balancing. Where necessary, you can use an external third party load balancer. Selection and configuration of an external load balancer is outside the scope of this specification. When an external third-party load balancer is used, the integration effort must include enough analysis to ensure all performance and resource utilization requirements are met.

Limits and requirements

- Stateful load balancing is not supported by MetalLB. An alternate load balancer implementation must be used if this is a requirement for workload CNFs.
- You must ensure that the external IP address is routable from clients to the host network for the cluster.

Engineering considerations

- MetalLB is used in BGP mode only for telco core use models.
- For telco core use models, MetalLB is supported only with the OVN-Kubernetes network provider used in local gateway mode. See **routingViaHost** in "Cluster Network Operator".
- BGP configuration in MetalLB is expected to vary depending on the requirements of the network and peers.
 - You can configure address pools with variations in addresses, aggregation length, auto assignment, and so on.
 - MetalLB uses BGP for announcing routes only. Only the **transmitInterval** and **minimumTtl** parameters are relevant in this mode. Other parameters in the BFD profile should remain close to the defaults as shorter values can lead to false negatives and affect performance.

Additional resources

- [When to use MetalLB](#)

3.7.3.3. SR-IOV

New in this release

- You can now create virtual functions for Mellanox NICs with the SR-IOV Network Operator when secure boot is enabled in the cluster host. Before you can create the virtual functions, you must first skip the firmware configuration for the Mellanox NIC and manually allocate the number of virtual functions in the firmware before switching the system to secure boot.

Description

SR-IOV enables physical functions (PFs) to be divided into multiple virtual functions (VFs). VFs can then be assigned to multiple pods to achieve higher throughput performance while keeping the pods isolated. The SR-IOV Network Operator provisions and manages SR-IOV CNI, network device plugin, and other components of the SR-IOV stack.

Limits and requirements

- Only certain network interfaces are supported. See "Supported devices" for more information.
- Enabling SR-IOV and IOMMU: the SR-IOV Network Operator automatically enables IOMMU on the kernel command line.
- SR-IOV VFs do not receive link state updates from the PF. If a link down detection is required, it must be done at the protocol level.
- **MultiNetworkPolicy** CRs can be applied to **netdevice** networks only. This is because the implementation uses iptables, which cannot manage vfio interfaces.

Engineering considerations

- SR-IOV interfaces in **vfio** mode are typically used to enable additional secondary networks for applications that require high throughput or low latency.
- The **SriovOperatorConfig** CR must be explicitly created. This CR is included in the reference configuration policies, which causes it to be created during initial deployment.
- NICs that do not support firmware updates with UEFI secure boot or kernel lockdown must be preconfigured with sufficient virtual functions (VFs) enabled to support the number of VFs required by the application workload. For Mellanox NICs, you must disable the Mellanox vendor plugin in the SR-IOV Network Operator. See "Configuring an SR-IOV network device" for more information.
- To change the MTU value of a VF after the pod has started, do not configure the **SriovNetworkNodePolicy** MTU field. Instead, use the Kubernetes NMState Operator to set the MTU of the related PF.

Additional resources

- [About Single Root I/O Virtualization \(SR-IOV\) hardware networks](#)
- [Supported devices](#)
- [Configuring the SR-IOV Network Operator on Mellanox cards when Secure Boot is enabled](#)

3.7.3.4. NMState Operator

New in this release

- No reference design updates in this release

Description

The Kubernetes NMState Operator provides a Kubernetes API for performing state-driven network configuration across cluster nodes. It enables network interface configurations, static IPs and DNS, VLANs, trunks, bonding, static routes, MTU, and enabling promiscuous mode on the secondary interfaces. The cluster nodes periodically report on the state of each node's network interfaces to the API server.

Limits and requirements

Not applicable

Engineering considerations

- Initial networking configuration is applied using **NMStateConfig** content in the installation CRs. The NMState Operator is used only when required for network updates.
- When SR-IOV virtual functions are used for host networking, the NMState Operator (via **nodeNetworkConfigurationPolicy** CRs) is used to configure VF interfaces, such as VLANs and MTU.

Additional resources

- [Kubernetes NMState Operator](#)

3.7.4. Logging

New in this release

- No reference design updates in this release

Description

The Cluster Logging Operator enables collection and shipping of logs off the node for remote archival and analysis. The reference configuration uses Kafka to ship audit and infrastructure logs to a remote archive.

Limits and requirements

Not applicable

Engineering considerations

- The impact of cluster CPU use is based on the number or size of logs generated and the amount of log filtering configured.
- The reference configuration does not include shipping of application logs. The inclusion of application logs in the configuration requires you to evaluate the application logging rate and have sufficient additional CPU resources allocated to the reserved set.

Additional resources

- [Logging 6.0](#)

3.7.5. Power Management

New in this release

- No reference design updates in this release

Description

Use the Performance Profile to configure clusters with high power mode, low power mode, or mixed mode. The choice of power mode depends on the characteristics of the workloads running on the cluster, particularly how sensitive they are to latency. Configure the maximum latency for a low-latency pod by using the per-pod power management C-states feature.

Limits and requirements

- Power configuration relies on appropriate BIOS configuration, for example, enabling C-states and P-states. Configuration varies between hardware vendors.

Engineering considerations

- Latency: To ensure that latency-sensitive workloads meet requirements, you require a high-power or a per-pod power management configuration. Per-pod power management is only available for Guaranteed QoS pods with dedicated pinned CPUs.

Additional resources

- [performance.openshift.io/v2 API reference](#)
- [Configuring power saving for nodes](#)

- Configuring power saving for nodes that run colocated high and low priority workloads

3.7.6. Storage

New in this release

- No reference design updates in this release

Description

Cloud native storage services can be provided by Red Hat OpenShift Data Foundation or other third-party solutions.

OpenShift Data Foundation is a Ceph-based software-defined storage solution for containers. It provides block storage, file system storage, and on-premise object storage, which can be dynamically provisioned for both persistent and non-persistent data requirements. Telco core applications require persistent storage.



NOTE

All storage data might not be encrypted in flight. To reduce risk, isolate the storage network from other cluster networks. The storage network must not be reachable, or routable, from other cluster networks. Only nodes directly attached to the storage network should be allowed to gain access to it.

Additional resources

- [Red Hat OpenShift Data Foundation](#)

3.7.6.1. Red Hat OpenShift Data Foundation

New in this release

- No reference design updates in this release

Description

Red Hat OpenShift Data Foundation is a software-defined storage service for containers. For telco core clusters, storage support is provided by OpenShift Data Foundation storage services running externally to the application workload cluster. OpenShift Data Foundation supports separation of storage traffic using secondary CNI networks.

Limits and requirements

- In an IPv4/IPv6 dual-stack networking environment, OpenShift Data Foundation uses IPv4 addressing. For more information, see [Network requirements](#).

Engineering considerations

- OpenShift Data Foundation network traffic should be isolated from other traffic on a dedicated network, for example, by using VLAN isolation.

3.7.6.2. Additional storage solutions

You can use other storage solutions to provide persistent storage for telco core clusters. The configuration and integration of these solutions is outside the scope of the reference design specifications (RDS).

Integration of the storage solution into the telco core cluster must include proper sizing and performance analysis to ensure the storage meets overall performance and resource usage requirements.

3.7.7. Telco core deployment components

The following sections describe the various OpenShift Container Platform components and configurations that you use to configure the hub cluster with Red Hat Advanced Cluster Management (RHACM).

3.7.7.1. Red Hat Advanced Cluster Management

New in this release

- Image-based installation for single-node OpenShift clusters is the recommended install methodology.

Description

Red Hat Advanced Cluster Management (RHACM) provides Multi Cluster Engine (MCE) installation and ongoing GitOps ZTP lifecycle management for deployed clusters. You manage cluster configuration and upgrades declaratively by applying **Policy** custom resources (CRs) to clusters during maintenance windows.

You apply policies with the RHACM policy controller as managed by Topology Aware Lifecycle Manager. Configuration, upgrades, and cluster status are managed through the policy controller.

When installing managed clusters, RHACM applies labels and initial ignition configuration to individual nodes in support of custom disk partitioning, allocation of roles, and allocation to machine config pools. You define these configurations with **SiteConfig** or **ClusterInstance** CRs.

Limits and requirements

- Hub cluster sizing is discussed in [Sizing your cluster](#).
- RHACM scaling limits are described in [Performance and Scalability](#).

Engineering considerations

- When managing multiple clusters with unique content per installation, site, or deployment, using RHACM hub templating is strongly recommended. RHACM hub templating allows you to apply a consistent set of policies to clusters while providing for unique values per installation.

Additional resources

- [Using GitOps ZTP to provision clusters at the network far edge](#)
- [Red Hat Advanced Cluster Management for Kubernetes](#)

3.7.7.2. Topology Aware Lifecycle Manager

New in this release

No reference design updates in this release.

Description

Topology Aware Lifecycle Manager is an Operator which runs only on the hub cluster. TALM manages how changes including cluster and Operator upgrades, configurations, and so on, are rolled out to managed clusters in the network. TALM has the following core features:

- Provides sequenced updates of cluster configurations and upgrades (OpenShift Container Platform and Operators) as defined by cluster policies.
- Provides for deferred application of cluster updates.
- Supports progressive rollout of policy updates to sets of clusters in user configurable batches.
- Allows for per-cluster actions by adding **ztp-done** or similar user-defined labels to clusters.

Limits and requirements

- Supports concurrent cluster deployments in batches of 400.

Engineering considerations

- Only policies with the **ran.openshift.io/ztp-deploy-wave** annotation are applied by TALM during initial cluster installation.
- Any policy can be remediated by TALM under control of a user created **ClusterGroupUpgrade** CR.

Additional resources

- [Updating managed clusters with the Topology Aware Lifecycle Manager](#)

3.7.7.3. GitOps Operator and GitOps ZTP plugins

New in this release

- No reference design updates in this release

Description

The GitOps Operator provides a GitOps driven infrastructure for managing cluster deployment and configuration. Cluster definitions and configuration are maintained in a Git repository.

ZTP plugins provide support for generating **Installation** CRs from **SiteConfig** CRs and automatically wrapping configuration CRs in policies based on RHACM **PolicyGenerator** CRs.

The SiteConfig Operator provides improved support for generation of **Installation** CRs from **ClusterInstance** CRs.



IMPORTANT

Where possible, use **ClusterInstance** CRs for cluster installation instead of the **SiteConfig** with GitOps ZTP plugin method.

You should structure the Git repository according to release version, with all necessary artifacts (**SiteConfig**, **ClusterInstance**, **PolicyGenerator**, and **PolicyGenTemplate**, and supporting reference CRs) included. This enables deploying and managing multiple versions of the OpenShift platform and configuration versions to clusters simultaneously and through upgrades.

The recommended Git structure keeps reference CRs in a directory separate from customer or partner provided content. This means that you can import reference updates by simply overwriting existing content. Customer or partner-supplied CRs can be provided in a parallel directory to the reference CRs for easy inclusion in the generated configuration policies.

Limits and requirements

- Each ArgoCD application supports up to 300 nodes. Multiple ArgoCD applications can be used to achieve the maximum number of clusters supported by a single hub cluster.
- The **SiteConfig** CR must use the **extraManifests.searchPaths** field to reference the reference manifests.



NOTE

Since OpenShift Container Platform 4.15, the **spec.extraManifestPath** field is deprecated.

Engineering considerations

- Set the **MachineConfigPool (mcp)** CR **paused** field to true during a cluster upgrade maintenance window and set the **maxUnavailable** field to the maximum tolerable value. This prevents multiple cluster node reboots during upgrade, which results in a shorter overall upgrade. When you unpause the **mcp** CR, all the configuration changes are applied with a single reboot.



NOTE

During installation, custom **mcp** CRs can be paused along with setting **maxUnavailable** to 100% to improve installation times.

- To avoid confusion or unintentional overwriting when updating content, you should use unique and distinguishable names for custom CRs in the **reference-crs/** directory under core-overlay and extra manifests in Git.
- The **SiteConfig** CR allows multiple extra-manifest paths. When file names overlap in multiple directory paths, the last file found in the directory order list takes precedence.

Additional resources

- [Preparing the GitOps ZTP site configuration repository for version independence](#)
- [Adding custom content to the GitOps ZTP pipeline](#)

3.7.7.4. Monitoring

New in this release

- No reference design updates in this release

Description

The Cluster Monitoring Operator (CMO) is included by default in OpenShift Container Platform and provides monitoring (metrics, dashboards, and alerting) for the platform components and optionally user projects. You can customize the default log retention period, custom alert rules, and so on. The default handling of pod CPU and memory metrics, based on upstream Kubernetes and cAdvisor, makes a tradeoff favoring stale data over metric accuracy. This leads to spikes in reporting, which can create false alerts, depending on the user-specified thresholds. OpenShift Container Platform supports an opt-in Dedicated Service Monitor feature that creates an additional set of pod CPU and memory metrics that do not suffer from this behavior. For more information, see [Dedicated Service Monitors - Questions and Answers \(Red Hat Knowledgebase\)](#).

In addition to the default configuration, the following metrics are expected to be configured for telco core clusters:

- Pod CPU and memory metrics and alerts for user workloads

Limits and requirements

- You must enable the Dedicated Service Monitor feature to represent pod metrics accurately.

Engineering considerations

- The Prometheus retention period is specified by the user. The value used is a tradeoff between operational requirements for maintaining historical data on the cluster against CPU and storage resources. Longer retention periods increase the need for storage and require additional CPU to manage data indexing.

Additional resources

- [About OpenShift Container Platform monitoring](#)

3.7.8. Scheduling

New in this release

- No reference design updates in this release

Description

The scheduler is a cluster-wide component responsible for selecting the correct node for a given workload. It is a core part of the platform and does not require any specific configuration in the common deployment scenarios. However, a few specific use cases are described in the following section.

NUMA-aware scheduling can be enabled through the NUMA Resources Operator. For more information, see "Scheduling NUMA-aware workloads".

Limits and requirements

- The default scheduler does not understand the NUMA locality of workloads. It only knows about the sum of all free resources on a worker node. This might cause workloads to be rejected when scheduled to a node with the topology manager policy set to **single-numa-node** or **restricted**. For more information, see "Topology Manager policies".

- For example, consider a pod requesting 6 CPUs that is scheduled to an empty node that has 4 CPUs per NUMA node. The total allocatable capacity of the node is 8 CPUs. The scheduler places the pod on the empty node. The node local admission fails, as there are only 4 CPUs available in each of the NUMA nodes.
- All clusters with multi-NUMA nodes are required to use the NUMA Resources Operator. See "Installing the NUMA Resources Operator" for more information. Use the **machineConfigPoolSelector** field in the **KubeletConfig** CR to select all nodes where NUMA aligned scheduling is required.
- All machine config pools must have consistent hardware configuration. For example, all nodes are expected to have the same NUMA zone count.

Engineering considerations

- Pods might require annotations for correct scheduling and isolation. For more information about annotations, see "CPU partitioning and performance tuning".
- You can configure SR-IOV virtual function NUMA affinity to be ignored during scheduling by using the **excludeTopology** field in **SriovNetworkNodePolicy** CR.

Additional resources

- [Installing the NUMA Resources Operator](#)
- [Scheduling NUMA-aware workloads](#)

Topology Manager policies

3.7.9. Node Configuration

New in this release

- No reference design updates in this release

Limits and requirements

- Analyze additional kernel modules to determine impact on CPU load, system performance, and ability to meet KPIs.

Table 3.1. Additional kernel modules

Feature	Description
---------	-------------

Feature	Description
Additional kernel modules	<p>Install the following kernel modules by using MachineConfig CRs to provide extended kernel functionality to CNFs.</p> <ul style="list-style-type: none"> ○ sctp ○ ip_gre ○ ip6_tables ○ ip6t_REJECT ○ ip6table_filter ○ ip6table_mangle ○ iptable_filter ○ iptable_mangle ○ iptable_nat ○ xt_multiport ○ xt_owner ○ xt_REDIRECT ○ xt_statistic ○ xt_TCPMSS
Container mount namespace hiding	<p>Reduce the frequency of kubelet housekeeping and eviction monitoring to reduce CPU usage. Creates a container mount namespace, visible to kubelet/CRI-O, to reduce system mount scanning overhead.</p>
Kdump enable	<p>Optional configuration (enabled by default)</p>

Additional resources

- [Automatic kernel crash dumps with kdump](#)
- [Optimizing CPU usage with mount namespace encapsulation](#)

3.7.10. Host firmware and boot loader configuration

New in this release

- No reference design updates in this release

Engineering considerations

- Enabling secure boot is the recommended configuration.



NOTE

When secure boot is enabled, only signed kernel modules are loaded by the kernel. Out-of-tree drivers are not supported.

3.7.11. Disconnected environment

New in this release

- No reference design updates in this release

Description

Telco core clusters are expected to be installed in networks without direct access to the internet. All container images needed to install, configure, and operate the cluster must be available in a disconnected registry. This includes OpenShift Container Platform images, Day 2 OLM Operator images, and application workload images. The use of a disconnected environment provides multiple benefits, including:

- Security – limiting access to the cluster
- Curated content – the registry is populated based on curated and approved updates for clusters

Limits and requirements

- A unique name is required for all custom **CatalogSource** resources. Do not reuse the default catalog names.

Engineering considerations

- A valid time source must be configured as part of cluster installation

Additional resources

- [About cluster updates in a disconnected environment](#)

3.7.12. Agent-based Installer

New in this release

- No reference design updates in this release

Description

Telco core clusters can be installed by using the Agent-based Installer. This method allows you to install OpenShift on bare-metal servers without requiring additional servers or VMs for managing the installation. The Agent-based Installer can be run on any system (for example, from a laptop) to generate an ISO installation image. The ISO is used as the installation media for the cluster supervisor nodes. Installation progress can be monitored using the ABI tool from any system with network connectivity to the supervisor node's API interfaces.

ABI supports the following:

- Installation from declarative CRs
- Installation in disconnected environments
- Installation with no additional supporting install or bastion servers required to complete the installation

Limits and requirements

- Disconnected installation requires a registry that is reachable from the installed host, with all required content mirrored in that registry.

Engineering considerations

- Networking configuration should be applied as NMState configuration during installation. Day 2 networking configuration using the NMState Operator is not supported.

Additional resources

- [Installing an OpenShift Container Platform cluster with the Agent-based Installer](#)

3.7.13. Security

New in this release

- [New knowledgebase article on creating custom node firewall rules](#)

Description

Telco customers are security conscious and require clusters to be hardened against multiple attack vectors. In OpenShift Container Platform, there is no single component or feature responsible for securing a cluster. Use the following security-oriented features and configurations to secure your clusters:

- **SecurityContextConstraints (SCC)**: All workload pods should be run with **restricted-v2** or **restricted** SCC.
- **Seccomp**: All pods should run with the **RuntimeDefault** (or stronger) seccomp profile.
- **Rootless DPDK pods**: Many user-plane networking (DPDK) CNFs require pods to run with root privileges. With this feature, a conformant DPDK pod can run without requiring root privileges. Rootless DPDK pods create a tap device in a rootless pod that injects traffic from a DPDK application to the kernel.
- **Storage**: The storage network should be isolated and non-routable to other cluster networks. See the "Storage" section for additional details.

Refer to [Custom nftables firewall rules in OpenShift](#) for a supported method of implementing custom nftables firewall rules in OpenShift cluster nodes. This article is intended for cluster administrators who are responsible for managing network security policies in OpenShift environments. It is crucial to carefully consider the operational implications before deploying this method, including:

- **Early application**: The rules are applied at boot time, before the network is fully operational. Ensure the rules don't inadvertently block essential services required during the boot process.

- **Risk of misconfiguration** Errors in your custom rules can lead to unintended consequences, potentially leading to performance impact or blocking legitimate traffic or isolating nodes. Thoroughly test your rules in a non-production environment before deploying them to your main cluster.
- **External endpoints:** OpenShift requires access to external endpoints to function. For more information about the firewall allowlist, see "Configuring your firewall for OpenShift Container Platform". Ensure that cluster nodes are permitted access to those endpoints.
- **Node reboot:** Unless node disruption policies are configured, applying the **MachineConfig** CR with the required firewall settings causes a node reboot. Be aware of this impact and schedule a maintenance window accordingly. For more information, see "Using node disruption policies to minimize disruption from machine config changes".



NOTE

Node disruption policies are available in OpenShift Container Platform 4.17 and later.

- **Network flow matrix** For more information about managing ingress traffic, see "OpenShift Container Platform network flow matrix". You can restrict ingress traffic to essential flows to improve network security. The matrix provides insights into base cluster services but excludes traffic generated by Day-2 Operators.
- **Cluster version updates and upgrades** Exercise caution when updating or upgrading OpenShift clusters. Recent changes to the platform's firewall requirements might require adjustments to network port permissions. Although the documentation provides guidelines, note that these requirements can evolve over time. To minimize disruptions, you should test any updates or upgrades in a staging environment before applying them in production. This helps you to identify and address potential compatibility issues related to firewall configuration changes.

Limits and requirements

- Rootless DPDK pods requires the following additional configuration:
 - Configure the **container_t** SELinux context for the tap plugin.
 - Enable the **container_use_devices** SELinux boolean for the cluster host.

Engineering considerations

- For rootless DPDK pod support, enable the SELinux **container_use_devices** boolean on the host to allow the tap device to be created. This introduces an acceptable security risk.

Additional resources

- [Configuring your firewall for OpenShift Container Platform](#)
- [OpenShift Container Platform network flow matrix](#)
- [Managing security context constraints](#)
- [Using node disruption policies to minimize disruption from machine config changes](#)

3.7.14. Scalability

New in this release

- No reference design updates in this release

Description

Scaling of workloads is described in "Application workloads".

Limits and requirements

- Cluster can scale to at least 120 nodes.

3.8. TELCO CORE REFERENCE CONFIGURATION CRS

Use the following custom resources (CRs) to configure and deploy OpenShift Container Platform clusters with the telco core profile. Use the CRs to form the common baseline used in all the specific use models unless otherwise indicated.

3.8.1. Extracting the telco core reference design configuration CRs

You can extract the complete set of custom resources (CRs) for the telco core profile from the **telco-core-rds-rhel9** container image. The container image has both the required CRs, and the optional CRs, for the telco core profile.

Prerequisites

- You have installed **podman**.

Procedure

- Extract the content from the **telco-core-rds-rhel9** container image by running the following commands:

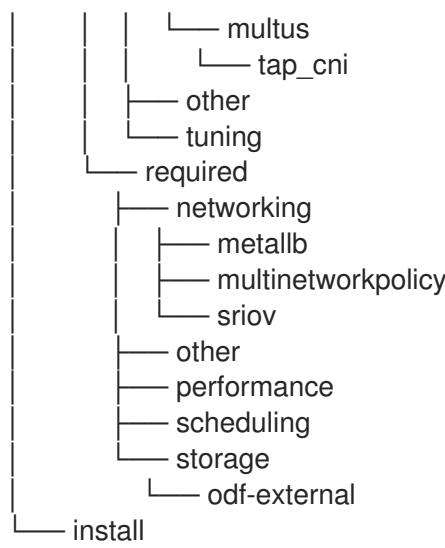
```
$ mkdir -p ./out
$ podman run -it registry.redhat.io/openshift4/openshift-telco-core-rds-rhel9:v4.18 | base64 -d | tar xv -C out
```

Verification

- The **out** directory has the following directory structure. You can view the telco core CRs in the **out/telco-core-rds/** directory.

Example output

```
out/
  └── telco-core-rds
      ├── configuration
      │   └── reference-crs
      │       ├── optional
      │       │   ├── logging
      │       │   └── networking
      └──
```



3.8.2. Comparing a cluster with the telco core reference configuration

After you deploy a telco core cluster, you can use the **cluster-compare** plugin to assess the cluster's compliance with the telco core reference design specifications (RDS). The **cluster-compare** plugin is an OpenShift CLI (**oc**) plugin. The plugin uses a telco core reference configuration to validate the cluster with the telco core custom resources (CRs).

The plugin-specific reference configuration for telco core is packaged in a container image with the telco core CRs.

For further information about the **cluster-compare** plugin, see "Understanding the cluster-compare plugin".

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have credentials to access the **registry.redhat.io** container image registry.
- You installed the **cluster-compare** plugin.

Procedure

1. Log on to the container image registry with your credentials by running the following command:

```
$ podman login registry.redhat.io
```

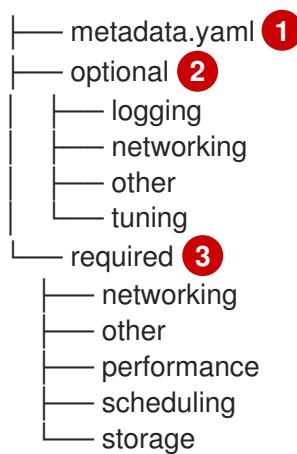
2. Extract the content from the **telco-core-rds-rhel9** container image by running the following commands:

```
$ mkdir -p ./out
```

```
$ podman run -it registry.redhat.io/openshift4/openshift-telco-core-rds-rhel9:v4.18 | base64 -d | tar xv -C out
```

You can view the reference configuration in the **reference-crs-kube-compare/** directory.

```
out/telco-core-rds/configuration/reference-crs-kube-compare/
```



① ① Configuration file for the reference configuration.

② Directory for optional templates.

③ Directory for required templates.

3. Compare the configuration for your cluster to the telco core reference configuration by running the following command:

```
$ oc cluster-compare -r out/telco-core-rds/configuration/reference-crs-kube-
compare/metadata.yaml
```

Example output

```
W1212 14:13:06.281590 36629 compare.go:425] Reference Contains Templates With
Types (kind) Not Supported By Cluster: BFDProfile, BGPAAdvertisement, BGPPeer,
ClusterLogForwarder, Community, IPAddressPool, MetallLB, MultiNetworkPolicy, NMState,
NUMAResourcesOperator, NUMAResourcesScheduler, NodeNetworkConfigurationPolicy,
SriovNetwork, SriovNetworkNodePolicy, SriovOperatorConfig, StorageCluster
```

...

Cluster CR: config.openshift.io/v1_OperatorHub_cluster ①

Reference File: required/other/operator-hub.yaml ②

Diff Output: diff -u -N /tmp/MERGED-2801470219/config-openshift-io-v1_operatorhub_cluster
/tmp/LIVE-2569768241/config-openshift-io-v1_operatorhub_cluster

```
--- /tmp/MERGED-2801470219/config-openshift-io-v1_operatorhub_cluster 2024-12-12
14:13:22.898756462 +0000
+++ /tmp/LIVE-2569768241/config-openshift-io-v1_operatorhub_cluster 2024-12-12
14:13:22.898756462 +0000
```

@@ -1,6 +1,6 @@

apiVersion: config.openshift.io/v1

kind: OperatorHub

metadata:

+ annotations: ③

+ include.release.openshift.io/hypershift: "true"

name: cluster

-spec:

- disableAllDefaultSources: true

Summary 4

CRs with diffs: 3/4 5

CRs in reference missing from the cluster: 22 6

other:

other:

Missing CRs: 7

- optional/other/control-plane-load-kernel-modules.yaml
- optional/other/worker-load-kernel-modules.yaml

required-networking:

networking-root:

Missing CRs:

- required/networking/nodeNetworkConfigurationPolicy.yaml

networking-sriov:

Missing CRs:

- required/networking/sriov/sriovNetwork.yaml
- required/networking/sriov/sriovNetworkNodePolicy.yaml
- required/networking/sriov/SriovOperatorConfig.yaml
- required/networking/sriov/SriovSubscription.yaml
- required/networking/sriov/SriovSubscriptionNS.yaml
- required/networking/sriov/SriovSubscriptionOperGroup.yaml

required-other:

scheduling:

Missing CRs:

- required/other/catalog-source.yaml
- required/other/icsp.yaml

required-performance:

performance:

Missing CRs:

- required/performance/PerformanceProfile.yaml

required-scheduling:

scheduling:

Missing CRs:

- required/scheduling/nrop.yaml
- required/scheduling/NROPSSubscription.yaml
- required/scheduling/NROPSSubscriptionNS.yaml
- required/scheduling/NROPSSubscriptionOperGroup.yaml
- required/scheduling/sched.yaml

required-storage:

storage-odf:

Missing CRs:

- required/storage/odf-external/01-rook-ceph-external-cluster-details.secret.yaml
- required/storage/odf-external/02-ocs-external-storagecluster.yaml
- required/storage/odf-external/odfNS.yaml
- required/storage/odf-external/odfOperGroup.yaml
- required/storage/odf-external/odfSubscription.yaml

No CRs are unmatched to reference CRs 8

Metadata Hash:

fe41066bac56517be02053d436c815661c9fa35eec5922af25a1be359818f297 9

No patched CRs 10

1 The CR under comparison. The plugin displays each CR with a difference from the corresponding template.

- 2** The template matching with the CR for comparison.
- 3** The output in Linux diff format shows the difference between the template and the cluster CR.
- 4** After the plugin reports the line diffs for each CR, the summary of differences are reported.
- 5** The number of CRs in the comparison with differences from the corresponding templates.
- 6** The number of CRs represented in the reference configuration, but missing from the live cluster.
- 7** The list of CRs represented in the reference configuration, but missing from the live cluster.
- 8** The CRs that did not match to a corresponding template in the reference configuration.
- 9** The metadata hash identifies the reference configuration.
- 10** The list of patched CRs.

Additional resources

- [Understanding the cluster-compare plugin](#)

3.8.3. Node configuration reference CRs

Table 3.2. Node configuration CRs

Component	Reference CR	Description	Optional
Additional kernel modules	control-plane-load-kernel-modules.yaml	Optional. Configures the kernel modules for control plane nodes.	No
Additional kernel modules	sctp_module_mc.yaml	Optional. Loads the SCTP kernel module in worker nodes.	No
Additional kernel modules	worker-load-kernel-modules.yaml	Optional. Configures kernel modules for worker nodes.	No
Container mount namespace hiding	mount_namespace_config_master.yaml	Configures a mount namespace for sharing container-specific mounts between kubelet and CRI-O on control plane nodes.	No

Component	Reference CR	Description	Optional
Container mount namespace hiding	mount_namespace_config_worker.yaml	Configures a mount namespace for sharing container-specific mounts between kubelet and CRI-O on worker nodes.	No
Kdump enable	kdump-master.yaml	Configures kdump crash reporting on master nodes.	No
Kdump enable	kdump-worker.yaml	Configures kdump crash reporting on worker nodes.	No

3.8.4. Resource tuning reference CRs

Table 3.3. Resource tuning CRs

Component	Reference CR	Description	Optional
System reserved capacity	control-plane-system-reserved.yaml	Optional. Configures kubelet, enabling auto-sizing reserved resources for the control plane node pool.	No

3.8.5. Networking reference CRs

Table 3.4. Networking CRs

Component	Reference CR	Description	Optional
Baseline	Network.yaml	Configures the default cluster network, specifying OVN Kubernetes settings like routing via the host. It also allows the definition of additional networks, including custom CNI configurations, and enables the use of MultiNetworkPolicy CRs for network policies across multiple networks.	No

Component	Reference CR	Description	Optional
Baseline	networkAttachmentDefinition.yaml	Optional. Defines a NetworkAttachmentDefinition resource specifying network configuration details such as node selector and CNI configuration.	Yes
Load Balancer	addr-pool.yaml	Configures MetalLB to manage a pool of IP addresses with auto-assign enabled for dynamic allocation of IPs from the specified range.	No
Load Balancer	bfd-profile.yaml	Configures bidirectional forwarding detection (BFD) with customized intervals, detection multiplier, and modes for quicker network fault detection and load balancing failover.	No
Load Balancer	bgp-advr.yaml	Defines a BGP advertisement resource for MetalLB, specifying how an IP address pool is advertised to BGP peers. This enables fine-grained control over traffic routing and announcements.	No
Load Balancer	bgp-peer.yaml	Defines a BGP peer in MetalLB, representing a BGP neighbor for dynamic routing.	No
Load Balancer	community.yaml	Defines a MetalLB community, which groups one or more BGP communities under a named resource. Communities can be applied to BGP advertisements to control routing policies and change traffic routing.	No

Component	Reference CR	Description	Optional
Load Balancer	metallb.yaml	Defines the MetalLB resource in the cluster.	No
Load Balancer	metallbNS.yaml	Defines the metallb-system namespace in the cluster.	No
Load Balancer	metallbOperGroup.yaml	Defines the Operator group for the MetalLB Operator.	No
Load Balancer	metallbSubscription.yaml	Creates a subscription resource for the metallb Operator with manual approval for install plans.	No
Multus - Tap CNI for rootless DPDK pods	mc_rootless_pods_s selinux.yaml	Configures a MachineConfig resource which sets an SELinux boolean for the tap CNI plugin on worker nodes.	Yes
NMState Operator	NMState.yaml	Defines an NMState resource that is used by the NMState Operator to manage node network configurations.	No
NMState Operator	NMStateNS.yaml	Creates the NMState Operator namespace.	No
NMState Operator	NMStateOperGroup.yaml	Creates the Operator group in the openshift-nmstate namespace, allowing the NMState Operator to watch and manage resources.	No

Component	Reference CR	Description	Optional
NMState Operator	NMStateSubscription.yaml	Creates a subscription for the NMState Operator, managed through OLM.	No
SR-IOV Network Operator	sriovNetwork.yaml	Defines an SR-IOV network specifying network capabilities, IP address management (ipam), and the associated network namespace and resource.	No
SR-IOV Network Operator	sriovNetworkNodePolicy.yaml	Configures network policies for SR-IOV devices on specific nodes, including customization of device selection, VF allocation (numVfs), node-specific settings (nodeSelector), and priorities.	No
SR-IOV Network Operator	SriovOperatorConfig.yaml	Configures various settings for the SR-IOV Operator, including enabling the injector and Operator webhook, disabling pod draining, and defining the node selector for the configuration daemon.	No
SR-IOV Network Operator	SriovSubscription.yaml	Creates a subscription for the SR-IOV Network Operator, managed through OLM.	No
SR-IOV Network Operator	SriovSubscriptionNS.yaml	Creates the SR-IOV Network Operator subscription namespace.	No
SR-IOV Network Operator	SriovSubscriptionOperatorGroup.yaml	Creates the Operator group for the SR-IOV Network Operator, allowing it to watch and manage resources in the target namespace.	No

3.8.6. Scheduling reference CRs

Table 3.5. Scheduling CRs

Component	Reference CR	Description	Optional
NUMA-aware scheduler	nrop.yaml	Enables the NUMA Resources Operator, aligning workloads with specific NUMA node configurations. Required for clusters with multi-NUMA nodes.	No
NUMA-aware scheduler	NROPSubscription.yaml	Creates a subscription for the NUMA Resources Operator, managed through OLM. Required for clusters with multi-NUMA nodes.	No
NUMA-aware scheduler	NROPSubscriptionNS.yaml	Creates the NUMA Resources Operator subscription namespace. Required for clusters with multi-NUMA nodes.	No
NUMA-aware scheduler	NROPSubscriptionOperatorGroup.yaml	Creates the Operator group in the numaresources-operator namespace, allowing the NUMA Resources Operator to watch and manage resources. Required for clusters with multi-NUMA nodes.	No
NUMA-aware scheduler	sched.yaml	Configures a topology-aware scheduler in the cluster that can handle NUMA aware scheduling of pods across nodes.	No
NUMA-aware scheduler	Scheduler.yaml	Configures control plane nodes as non-schedulable for workloads.	No

3.8.7. Storage reference CRs

Table 3.6. Storage CRs

Component	Reference CR	Description	Optional
External ODF configuration	01-rook-ceph-external-cluster-details.secret.yaml	Defines a Secret resource containing base64-encoded configuration data for an external Ceph cluster in the openshift-storage namespace.	No
External ODF configuration	02-ocs-external-storagecluster.yaml	Defines an OpenShift Container Storage (OCS) storage resource which configures the cluster to use an external storage back end.	No
External ODF configuration	odfNS.yaml	Creates the monitored openshift-storage namespace for the OpenShift Data Foundation Operator.	No
External ODF configuration	odfOperGroup.yaml	Creates the Operator group in the openshift-storage namespace, allowing the OpenShift Data Foundation Operator to watch and manage resources.	No
External ODF configuration	odfSubscription.yaml	Creates the subscription for the OpenShift Data Foundation Operator in the openshift-storage namespace.	No

3.9. TELCO CORE REFERENCE CONFIGURATION SOFTWARE SPECIFICATIONS

The Red Hat telco core 4.18 solution has been validated using the following Red Hat software products for OpenShift Container Platform clusters.

Table 3.7. Telco core cluster validated software components

Component	Software version
Red Hat Advanced Cluster Management (RHACM)	2.12 ¹

Component	Software version
Cluster Logging Operator	6.1 ²
OpenShift Data Foundation	4.18
SR-IOV Network Operator	4.18
MetalLB	4.18
NMState Operator	4.18
NUMA-aware scheduler	4.18

[1] This table will be updated when the aligned RHACM version 2.13 is released.

[2] This table will be updated when the aligned Cluster Logging Operator 6.2 is released.

CHAPTER 4. TELCO RAN DU REFERENCE DESIGN SPECIFICATIONS

The telco RAN DU reference design specifications (RDS) describes the configuration for clusters running on commodity hardware to host 5G workloads in the Radio Access Network (RAN). It captures the recommended, tested, and supported configurations to get reliable and repeatable performance for a cluster running the telco RAN DU profile.

4.1. REFERENCE DESIGN SPECIFICATIONS FOR TELCO RAN DU 5G DEPLOYMENTS

Red Hat and certified partners offer deep technical expertise and support for networking and operational capabilities required to run telco applications on OpenShift Container Platform 4.18 clusters.

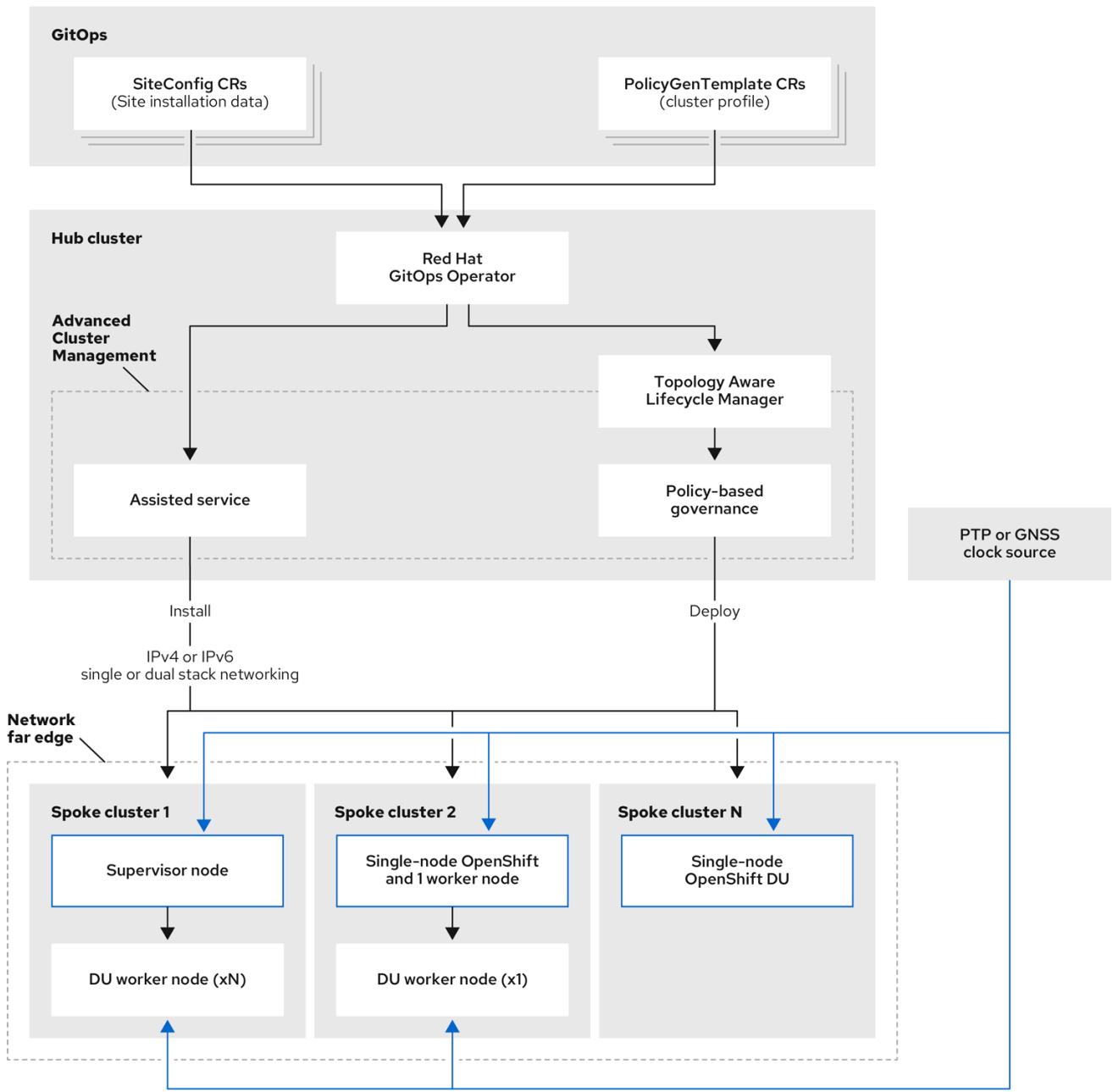
Red Hat's telco partners require a well-integrated, well-tested, and stable environment that can be replicated at scale for enterprise 5G solutions. The telco core and RAN DU reference design specifications (RDS) outline the recommended solution architecture based on a specific version of OpenShift Container Platform. Each RDS describes a tested and validated platform configuration for telco core and RAN DU use models. The RDS ensures an optimal experience when running your applications by defining the set of critical KPIs for telco 5G core and RAN DU. Following the RDS minimizes high severity escalations and improves application stability.

5G use cases are evolving and your workloads are continually changing. Red Hat is committed to iterating over the telco core and RAN DU RDS to support evolving requirements based on customer and partner feedback.

The reference configuration includes the configuration of the far edge clusters and hub cluster components.

The reference configurations in this document are deployed using a centrally managed hub cluster infrastructure as shown in the following image.

Figure 4.1. Telco RAN DU deployment architecture



474_OpenShift_1023

4.2. REFERENCE DESIGN SCOPE

The telco core, telco RAN and telco hub reference design specifications (RDS) capture the recommended, tested, and supported configurations to get reliable and repeatable performance for clusters running the telco core and telco RAN profiles.

Each RDS includes the released features and supported configurations that are engineered and validated for clusters to run the individual profiles. The configurations provide a baseline OpenShift Container Platform installation that meets feature and KPI targets. Each RDS also describes expected variations for each individual configuration. Validation of each RDS includes many long duration and at-scale tests.

**NOTE**

The validated reference configurations are updated for each major Y-stream release of OpenShift Container Platform. Z-stream patch releases are periodically re-tested against the reference configurations.

4.3. DEVIATIONS FROM THE REFERENCE DESIGN

Deviating from the validated telco core, telco RAN DU, and telco hub reference design specifications (RDS) can have significant impact beyond the specific component or feature that you change. Deviations require analysis and engineering in the context of the complete solution.

**IMPORTANT**

All deviations from the RDS should be analyzed and documented with clear action tracking information. Due diligence is expected from partners to understand how to bring deviations into line with the reference design. This might require partners to provide additional resources to engage with Red Hat to work towards enabling their use case to achieve a best in class outcome with the platform. This is critical for the supportability of the solution and ensuring alignment across Red Hat and with partners.

Deviation from the RDS can have some or all of the following consequences:

- It can take longer to resolve issues.
- There is a risk of missing project service-level agreements (SLAs), project deadlines, end provider performance requirements, and so on.
- Unapproved deviations may require escalation at executive levels.

**NOTE**

Red Hat prioritizes the servicing of requests for deviations based on partner engagement priorities.

4.4. ENGINEERING CONSIDERATIONS FOR THE RAN DU USE MODEL

The RAN DU use model configures an OpenShift Container Platform cluster running on commodity hardware for hosting RAN distributed unit (DU) workloads. Model and system level considerations are described below. Specific limits, requirements and engineering considerations for individual components are detailed in later sections.

**NOTE**

For details of the RAN DU KPI test results, see the [Telco RAN DU reference design specification KPI test results for OpenShift 4.18](#). This information is only available to customers and partners.

Workloads

- DU workloads are described in "Telco RAN DU application workloads".
- DU worker nodes are Intel 3rd Generation Xeon (IceLake) 2.20 GHz or better with host firmware tuned for maximum performance.

Resources

The maximum number of running pods in the system, inclusive of application workload and OpenShift Container Platform pods, is 120.

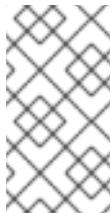
Resource utilization

OpenShift Container Platform resource utilization varies depending on many factors such as the following application workload characteristics:

- Pod count
- Type and frequency of probes
- Messaging rates on the primary or secondary CNI with kernel networking
- API access rate
- Logging rates
- Storage IOPS

Resource utilization is measured for clusters configured as follows:

1. The cluster is a single host with single-node OpenShift installed.
2. The cluster runs the representative application workload described in "Reference application workload characteristics".
3. The cluster is managed under the constraints detailed in "Hub cluster management characteristics".
4. Components noted as "optional" in the use model configuration are not included.



NOTE

Configuration outside the scope of the RAN DU RDS that do not meet these criteria requires additional analysis to determine the impact on resource utilization and ability to meet KPI targets. You might need to allocate additional cluster resources to meet these requirements.

Reference application workload characteristics

1. Uses 15 pods and 30 containers for the vRAN application including its management and control functions
2. Uses an average of 2 **ConfigMap** and 4 **Secret** CRs per pod
3. Uses a maximum of 10 exec probes with a frequency of not less than 10 seconds
4. Incremental application load on the kube-apiserver is less than or equal to 10% of the cluster platform usage

**NOTE**

You can extract CPU load can from the platform metrics. For example:

```
$
query=avg_over_time(pod:container_cpu_usage:sum{namespace="openshift-kube-apiserver"}[30m])
```

5. Application logs are not collected by the platform log collector
6. Aggregate traffic on the primary CNI is less than 8 Mbps

Hub cluster management characteristics

RHACM is the recommended cluster management solution and is configured to these limits:

1. Use a maximum of 5 RHACM configuration policies with a compliant evaluation interval of not less than 10 minutes.
2. Use a minimal number (up to 10) of managed cluster templates in cluster policies. Use hub-side templating.
3. Disable RHACM addons with the exception of the **policyController** and configure observability with the default configuration.

The following table describes resource utilization under reference application load.

Table 4.1. Resource utilization under reference application load

Metric	Limits	Notes
OpenShift platform CPU usage	Less than 4000mc – 2 cores (4HT)	Platform CPU is pinned to reserved cores, including both hyper-threads of each reserved core. The system is engineered to 3 CPUs (3000mc) at steady-state to allow for periodic system tasks and spikes.
OpenShift Platform memory	Less than 16G	

4.5. TELCO RAN DU APPLICATION WORKLOADS

Develop RAN DU applications that are subject to the following requirements and limitations.

Description and limits

- Develop cloud-native network functions (CNFs) that conform to the latest version of [Red Hat best practices for Kubernetes](#).
- Use SR-IOV for high performance networking.
- Use exec probes sparingly and only when no other suitable options are available.

- Do not use exec probes if a CNF uses CPU pinning. Use other probe implementations, for example, **httpGet** or **tcpSocket**.
- When you need to use exec probes, limit the exec probe frequency and quantity. The maximum number of exec probes must be kept below 10, and frequency must not be set to less than 10 seconds. Exec probes cause much higher CPU usage on management cores compared to other probe types because they require process forking.

**NOTE**

Startup probes require minimal resources during steady-state operation. The limitation on exec probes applies primarily to liveness and readiness probes.

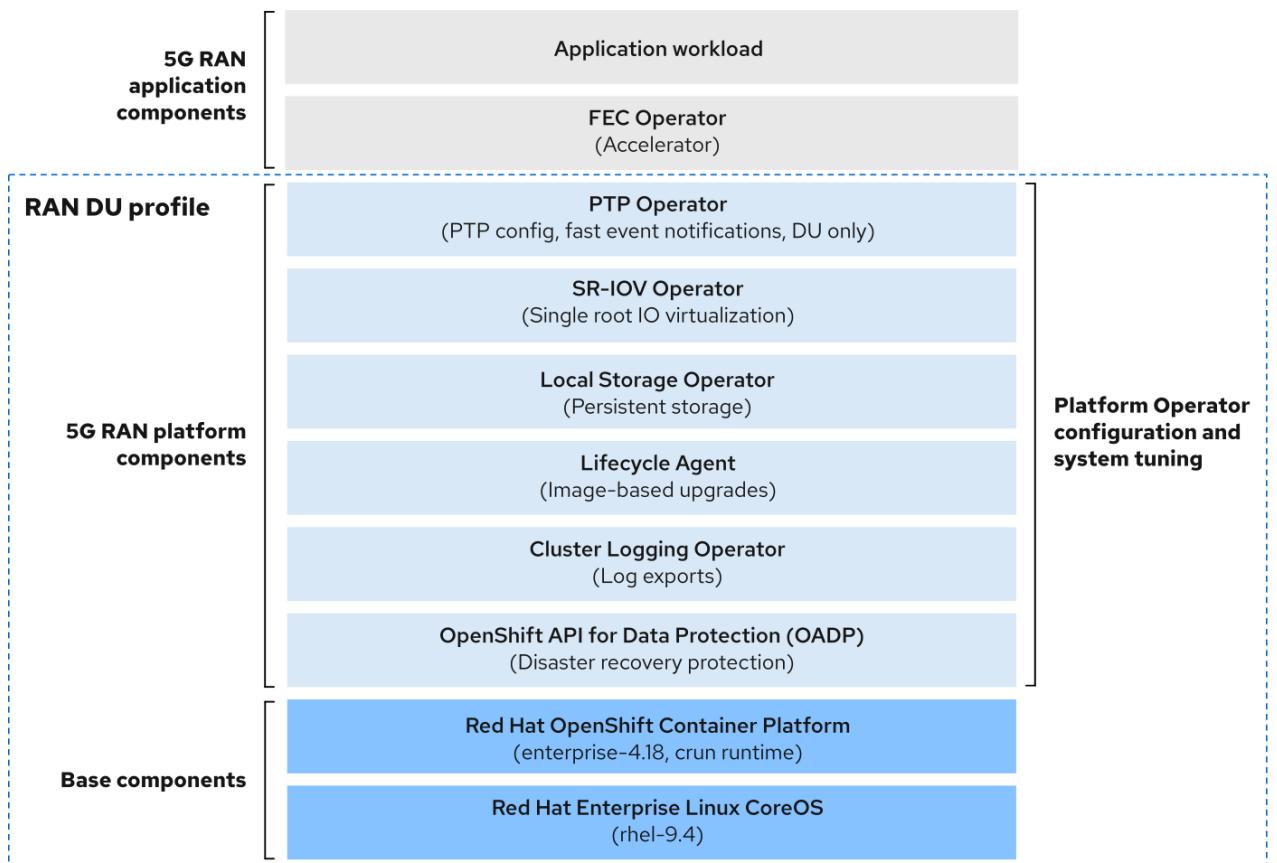
**NOTE**

A test workload that conforms to the dimensions of the reference DU application workload described in this specification can be found at [openshift-kni/du-test-workloads](#).

4.6. TELCO RAN DU REFERENCE DESIGN COMPONENTS

The following sections describe the various OpenShift Container Platform components and configurations that you use to configure and deploy clusters to run RAN DU workloads.

Figure 4.2. Telco RAN DU reference design components





NOTE

Ensure that additional components you include that are not specified in the telco RAN DU profile do not affect the CPU resources allocated to workload applications.



IMPORTANT

Out of tree drivers are not supported. 5G RAN application components are not included in the RAN DU profile and must be engineered against resources (CPU) allocated to applications.

4.6.1. Host firmware tuning

New in this release

- No reference design updates in this release

Description

Tune host firmware settings for optimal performance during initial cluster deployment. For more information, see "Recommended single-node OpenShift cluster configuration for vDU application workloads". Apply tuning settings in the host firmware during initial deployment. See "Managing host firmware settings with GitOps ZTP" for more information. The managed cluster host firmware settings are available on the hub cluster as individual **BareMetalHost** custom resources (CRs) that are created when you deploy the managed cluster with the **ClusterInstance** CR and GitOps ZTP.



NOTE

Create the **ClusterInstance** CR based on the provided reference [example-sno.yaml](#) CR.

Limits and requirements

- You must enable Hyper-Threading in the host firmware settings

Engineering considerations

- Tune all firmware settings for maximum performance.
- All settings are expected to be for maximum performance unless tuned for power savings.
- You can tune host firmware for power savings at the expense of performance as required.
- Enable secure boot. When secure boot is enabled, only signed kernel modules are loaded by the kernel. Out-of-tree drivers are not supported.

Additional resources

- [Managing host firmware settings with GitOps ZTP](#)
- [Configuring host firmware for low latency and high performance](#)
- [Provisioning real-time and low latency workloads](#)

4.6.2. CPU partitioning and performance tuning

New in this release

- No reference design updates in this release

Description

The RAN DU use model includes cluster performance tuning via **PerformanceProfile** CRs for low-latency performance. The **PerformanceProfile** CRs are reconciled by the Node Tuning Operator. The RAN DU use case requires the cluster to be tuned for low-latency performance. For more details about node tuning with the **PerformanceProfile** CR, see "Tuning nodes for low latency with the performance profile".

Limits and requirements

The Node Tuning Operator uses the **PerformanceProfile** CR to configure the cluster. You need to configure the following settings in the telco RAN DU profile **PerformanceProfile** CR:

- Set a reserved **cpuset** of 4 or more, equating to 4 hyper-threads (2 cores) for either of the following CPUs:
 - Intel 3rd Generation Xeon (IceLake) 2.20 GHz or better CPUs with host firmware tuned for maximum performance
 - AMD EPYC Zen 4 CPUs (Genoa, Bergamo, or newer)



NOTE

AMD EPYC Zen 4 CPUs (Genoa, Bergamo, or newer) are fully supported. Power consumption evaluations are ongoing. It is recommended to evaluate features, such as per-pod power management, to determine any potential impact on performance.

- Set the reserved **cpuset** to include both hyper-thread siblings for each included core. Unreserved cores are available as allocatable CPU for scheduling workloads.
- Ensure that hyper-thread siblings are not split across reserved and isolated cores.
- Ensure that reserved and isolated CPUs include all the threads for all cores in the CPU.
- Include Core 0 for each NUMA node in the reserved CPU set.
- Set the huge page size to 1G.
- Only pin OpenShift Container Platform pods which are by default configured as part of the management workload partition to reserved cores.

Engineering considerations

- Meeting the full performance metrics requires use of the RT kernel. If required, you can use the non-RT kernel with corresponding impact to performance.
- The number of hugepages you configure depends on application workload requirements. Variation in this parameter is expected and allowed.

- Variation is expected in the configuration of reserved and isolated CPU sets based on selected hardware and additional components in use on the system. The variation must still meet the specified limits.
- Hardware without IRQ affinity support affects isolated CPUs. To ensure that pods with guaranteed whole CPU QoS have full use of allocated CPUs, all hardware in the server must support IRQ affinity.
- When workload partitioning is enabled by setting **cpuPartitioningMode** to **AllNodes** during deployment, you must allocate enough CPUs to support the operating system, interrupts, and OpenShift Container Platform pods in the **PerformanceProfile** CR.

Additional resources

- [Finding the effective IRQ affinity setting for a node](#)

4.6.3. PTP Operator

New in this release

- No reference design updates in this release

Description

Configure PTP in cluster nodes with **PTPConfig** CRs for the RAN DU use case with features like Grandmaster clock (T-GM) support via GPS, ordinary clock (OC), boundary clocks (T-BC), dual boundary clocks, high availability (HA), and optional fast event notification over HTTP. PTP ensures precise timing and reliability in the RAN environment.

Limits and requirements

- Limited to two boundary clocks for nodes with dual NICs and HA
- Limited to two Westport channel NIC configurations for T-GM

Engineering considerations

- RAN DU RDS configurations are provided for ordinary clocks, boundary clocks, grandmaster clocks, and highly available dual NIC boundary clocks.
- PTP fast event notifications use **ConfigMap** CRs to persist subscriber details.
- Hierarchical event subscription as described in the O-RAN specification is not supported for PTP events.
- Use the PTP fast events REST API v2. The PTP fast events REST API v1 is deprecated. The REST API v2 is O-RAN Release 3 compliant.

4.6.4. SR-IOV Operator

New in this release

- No reference design updates in this release

Description

The SR-IOV Operator provisions and configures the SR-IOV CNI and device plugins. Both **netdevice** (kernel VFs) and **vfio** (DPDK) devices are supported and applicable to the RAN DU use models.

Limits and requirements

- Use devices that are supported for OpenShift Container Platform. See "Supported devices".
- SR-IOV and IOMMU enablement in host firmware settings: The SR-IOV Network Operator automatically enables IOMMU on the kernel command line.
- SR-IOV VFs do not receive link state updates from the PF. If link down detection is required you must configure this at the protocol level.

Engineering considerations

- SR-IOV interfaces with the **vfio** driver type are typically used to enable additional secondary networks for applications that require high throughput or low latency.
- Customer variation on the configuration and number of **SriovNetwork** and **SriovNetworkNodePolicy** custom resources (CRs) is expected.
- IOMMU kernel command-line settings are applied with a **MachineConfig** CR at install time. This ensures that the **SriovOperator** CR does not cause a reboot of the node when adding them.
- SR-IOV support for draining nodes in parallel is not applicable in a single-node OpenShift cluster.
- You must include the **SriovOperatorConfig** CR in your deployment; the CR is not created automatically. This CR is included in the reference configuration policies which are applied during initial deployment.
- In scenarios where you pin or restrict workloads to specific nodes, the SR-IOV parallel node drain feature will not result in the rescheduling of pods. In these scenarios, the SR-IOV Operator disables the parallel node drain functionality.
- NICs which do not support firmware updates under secure boot or kernel lockdown must be pre-configured with sufficient virtual functions (VFs) to support the number of VFs needed by the application workload. For Mellanox NICs, the Mellanox vendor plugin must be disabled in the SR-IOV Network Operator. For more information, see "Configuring an SR-IOV network device".
- To change the MTU value of a virtual function after the pod has started, do not configure the MTU field in the **SriovNetworkNodePolicy** CR. Instead, configure the Network Manager or use a custom systemd script to set the MTU of the physical function to an appropriate value. For example:

```
# ip link set dev <physical_function> mtu 9000
```

Additional resources

- [Supported devices](#)
- [Configuring QinQ support for SR-IOV enabled workloads](#)

4.6.5. Logging

New in this release

- No reference design updates in this release

Description

Use logging to collect logs from the far edge node for remote analysis. The recommended log collector is Vector.

Engineering considerations

- Handling logs beyond the infrastructure and audit logs, for example, from the application workload requires additional CPU and network bandwidth based on additional logging rate.
- As of OpenShift Container Platform 4.14, Vector is the reference log collector. Use of fluentd in the RAN use models is deprecated.

Additional resources

- [Logging 6.0](#)

4.6.6. SRIOV-FEC Operator

New in this release

- No reference design updates in this release

Description

SRIOV-FEC Operator is an optional 3rd party Certified Operator supporting FEC accelerator hardware.

Limits and requirements

- Starting with FEC Operator v2.7.0:
 - Secure boot is supported
 - **vfio** drivers for PFs require the usage of a **vfio-token** that is injected into the pods. Applications in the pod can pass the VF token to DPDK by using EAL parameter **--vfio-vf-token**.

Engineering considerations

- The SRIOV-FEC Operator uses CPU cores from the isolated CPU set.
- You can validate FEC readiness as part of the pre-checks for application deployment, for example, by extending the validation policy.

Additional resources

- [SRIOV-FEC Operator for Intel® vRAN Dedicated Accelerator manager container](#)

4.6.7. Lifecycle Agent

New in this release

- No reference design updates in this release

Description

The Lifecycle Agent provides local lifecycle management services for single-node OpenShift clusters.

Limits and requirements

- The Lifecycle Agent is not applicable in multi-node clusters or single-node OpenShift clusters with an additional worker.
- The Lifecycle Agent requires a persistent volume that you create when installing the cluster. For descriptions of partition requirements, see "Configuring a shared container directory between ostree stateroots when using GitOps ZTP".

Additional resources

- [Understanding the image-based upgrade for single-node OpenShift clusters](#)
- [Configuring a shared container directory between ostree stateroots when using GitOps ZTP](#)

4.6.8. Local Storage Operator

New in this release

- No reference design updates in this release

Description

You can create persistent volumes that can be used as **PVC** resources by applications with the Local Storage Operator. The number and type of **PV** resources that you create depends on your requirements.

Engineering considerations

- Create backing storage for **PV** CRs before creating the **PV**. This can be a partition, a local volume, LVM volume, or full disk.
- Refer to the device listing in **LocalVolume** CRs by the hardware path used to access each device to ensure correct allocation of disks and partitions, for example, `/dev/disk/by-path/<id>`. Logical names (for example, `/dev/sda`) are not guaranteed to be consistent across node reboots.

4.6.9. Logical Volume Manager Storage

New in this release

- No reference design updates in this release

Description

Logical Volume Manager (LVM) Storage is an optional component. It provides dynamic provisioning of both block and file storage by creating logical volumes from local devices that can be consumed as persistent volume claim (PVC) resources by applications. Volume expansion and snapshots are also

possible. An example configuration is provided in the RDS with the **StorageLVMCluster.yaml** file.

Limits and requirements

- In single-node OpenShift clusters, persistent storage must be provided by either LVM Storage or local storage, not both.
- Volume snapshots are excluded from the reference configuration.

Engineering considerations

- LVM Storage can be used as the local storage implementation for the RAN DU use case. When LVM Storage is used as the storage solution, it replaces the Local Storage Operator, and the CPU required is assigned to the management partition as platform overhead. The reference configuration must include one of these storage solutions but not both.
- Ensure that sufficient disks or partitions are available for storage requirements.

4.6.10. Workload partitioning

New in this release

- No reference design updates in this release

Description

Workload partitioning pins OpenShift Container Platform and Day 2 Operator pods that are part of the DU profile to the reserved CPU set and removes the reserved CPU from node accounting. This leaves all unreserved CPU cores available for user workloads. This leaves all non-reserved CPU cores available for user workloads. Workload partitioning is enabled through a capability set in installation parameters: **cpuPartitioningMode: AllNodes**. The set of management partition cores are set with the reserved CPU set that you configure in the **PerformanceProfile** CR.

Limits and requirements

- **Namespace** and **Pod** CRs must be annotated to allow the pod to be applied to the management partition
- Pods with CPU limits cannot be allocated to the partition. This is because mutation can change the pod QoS.
- For more information about the minimum number of CPUs that can be allocated to the management partition, see "Node Tuning Operator".

Engineering considerations

- Workload partitioning pins all management pods to reserved cores. A sufficient number of cores must be allocated to the reserved set to account for operating system, management pods, and expected spikes in CPU use that occur when the workload starts, the node reboots, or other system events happen.

Additional resources

- [Workload partitioning](#)

4.6.11. Cluster tuning

New in this release

- No reference design updates in this release

Description

See "Cluster capabilities" for a full list of components that can be disabled by using the cluster capabilities feature.

Limits and requirements

- Cluster capabilities are not available for installer-provisioned installation methods.

Engineering considerations

- In clusters running OpenShift Container Platform 4.16 and later, the cluster does not automatically revert to cgroup v1 when a **PerformanceProfile** is applied. If workloads running on the cluster require cgroup v1, the cluster must be configured for cgroup v1. For more information, see "Enabling Linux control group version 1 (cgroup v1)". You should make this configuration as part of the initial cluster deployment.



NOTE

Support for cgroup v1 is planned for removal in OpenShift Container Platform 4.19. Clusters running cgroup v1 must transition to cgroup v2.

The following table lists the required platform tuning configurations:

Table 4.2. Cluster capabilities configurations

Feature	Description
Remove optional cluster capabilities	<p>Reduce the OpenShift Container Platform footprint by disabling optional cluster Operators on single-node OpenShift clusters only.</p> <ul style="list-style-type: none">• Remove all optional Operators except the Node Tuning Operator, Operator Lifecycle Manager, and the Ingress Operator.

Feature	Description
Configure cluster monitoring	<p>Configure the monitoring stack for reduced footprint by doing the following:</p> <ul style="list-style-type: none"> ● Disable the local alertmanager and telemeter components. ● If you use RHACM observability, the CR must be augmented with appropriate additionalAlertManagerConfigs CRs to forward alerts to the hub cluster. ● Reduce the Prometheus retention period to 24h.  <p>NOTE</p> <p>The RHACM hub cluster aggregates managed cluster metrics.</p>
Disable networking diagnostics	Disable networking diagnostics for single-node OpenShift because they are not required.
Configure a single OperatorHub catalog source	Configure the cluster to use a single catalog source that contains only the Operators required for a RAN DU deployment. Each catalog source increases the CPU use on the cluster. Using a single CatalogSource fits within the platform CPU budget.
Disable the Console Operator	If the cluster was deployed with the console disabled, the Console CR (ConsoleOperatorDisable.yaml) is not needed. If the cluster was deployed with the console enabled, you must apply the Console CR.

Additional resources

- [Cluster capabilities](#)

4.6.12. Machine configuration

New in this release

- No reference design updates in this release

Limits and requirements

The CRI-O wipe disable **MachineConfig** CR assumes that images on disk are static other than during scheduled maintenance in defined maintenance windows. To ensure the images are static, do not set the pod **imagePullPolicy** field to **Always**.

Table 4.3. Machine configuration options

Feature	Description
Container Runtime	Sets the container runtime to crun for all node roles.
Kubelet config and container mount namespace hiding	Reduces the frequency of kubelet housekeeping and eviction monitoring, which reduces CPU usage
SCTP	Optional configuration (enabled by default)
Kdump	Optional configuration (enabled by default) Enables kdump to capture debug information when a kernel panic occurs. The reference CRs that enable kdump have an increased memory reservation based on the set of drivers and kernel modules included in the reference configuration.
CRI-O wipe disable	Disables automatic wiping of the CRI-O image cache after unclean shutdown
SR-IOV-related kernel arguments	Include additional SR-IOV-related arguments in the kernel command line
Set RCU Normal	Systemd service that sets rcu_normal after the system finishes startup
One-shot time sync	Runs a one-time NTP system time synchronization job for control plane or worker nodes.

Additional resources

- [Recommended single-node OpenShift cluster configuration for vDU application workloads](#) .

4.7. TELCO RAN DU DEPLOYMENT COMPONENTS

The following sections describe the various OpenShift Container Platform components and configurations that you use to configure the hub cluster with RHACM.

4.7.1. Red Hat Advanced Cluster Management

New in this release

- No reference design updates in this release

Description

RHACM provides Multi Cluster Engine (MCE) installation and ongoing lifecycle management functionality for deployed clusters. You manage cluster configuration and upgrades declaratively by applying **Policy** custom resources (CRs) to clusters during maintenance windows.

RHACM provides the following functionality:

- Zero touch provisioning (ZTP) of clusters using the MCE component in RHACM.
- Configuration, upgrades, and cluster status through the RHACM policy controller.
- During managed cluster installation, RHACM can apply labels to individual nodes as configured through the **ClusterInstance** CR.

The recommended method for single-node OpenShift cluster installation is the image-based installer method available in MCE using the **ClusterInstance** CR for cluster definition.

The recommended method for single-node OpenShift cluster upgrade is the image-based upgrade method.

Limits and requirements

- A single hub cluster supports up to 3500 deployed single-node OpenShift clusters with 5 **Policy** CRs bound to each cluster.

Engineering considerations

- Use RHACM policy hub-side templating to better scale cluster configuration. You can significantly reduce the number of policies by using a single group policy or small number of general group policies where the group and per-cluster values are substituted into templates.
- Cluster specific configuration: managed clusters typically have some number of configuration values that are specific to the individual cluster. These configurations should be managed using RHACM policy hub-side templating with values pulled from **ConfigMap** CRs based on the cluster name.
- To save CPU resources on managed clusters, policies that apply static configurations should be unbound from managed clusters after GitOps ZTP installation of the cluster.

Additional resources

- [Using GitOps ZTP to provision clusters at the network far edge](#)
- [Red Hat Advanced Cluster Management for Kubernetes](#)

4.7.2. SiteConfig Operator

New in this release

- No RDS updates in this release

Description

The SiteConfig Operator is a template-driven solution designed to provision clusters through various installation methods. It introduces the unified **ClusterInstance** API, which replaces the deprecated **SiteConfig** API. By leveraging the **ClusterInstance** API, the SiteConfig Operator improves cluster provisioning by providing the following:

- Better isolation of definitions from installation methods
- Unification of Git and non-Git workflows

- Consistent APIs across installation methods
- Enhanced scalability
- Increased flexibility with custom installation templates
- Valuable insights for troubleshooting deployment issues

The SiteConfig Operator provides validated default installation templates to facilitate cluster deployment through both the Assisted Installer and Image-based Installer provisioning methods:

- **Assisted Installer** automates the deployment of OpenShift Container Platform clusters by leveraging predefined configurations and validated host setups. It ensures that the target infrastructure meets OpenShift Container Platform requirements. The Assisted Installer streamlines the installation process while minimizing time and complexity compared to manual setup.
- **Image-based Installer** expedites the deployment of single-node OpenShift clusters by utilizing preconfigured and validated OpenShift Container Platform seed images. Seed images are preinstalled on target hosts, enabling rapid reconfiguration and deployment. The Image-based Installer is particularly well-suited for remote or disconnected environments, because it simplifies the cluster creation process and significantly reduces deployment time.

Limits and requirements

- A single hub cluster supports up to 3500 deployed single-node OpenShift clusters.

4.7.3. Topology Aware Lifecycle Manager

New in this release

- No reference design updates in this release

Description

Topology Aware Lifecycle Manager is an Operator that runs only on the hub cluster for managing how changes like cluster upgrades, Operator upgrades, and cluster configuration are rolled out to the network. TALM supports the following features:

- Progressive rollout of policy updates to fleets of clusters in user configurable batches.
- Per-cluster actions add **ztp-done** labels or other user-configurable labels following configuration changes to managed clusters.
- Precaching of single-node OpenShift clusters images: TALM supports optional pre-caching of OpenShift, OLM Operator, and additional user images to single-node OpenShift clusters before initiating an upgrade. The precaching feature is not applicable when using the recommended image-based upgrade method for upgrading single-node OpenShift clusters.
 - Specifying optional pre-caching configurations with **PreCachingConfig** CRs. Review the [sample reference PreCachingConfig CR](#) for more information.
 - Excluding unused images with configurable filtering.
 - Enabling before and after pre-caching storage space validations with configurable space-required parameters.

Limits and requirements

- Supports concurrent cluster deployment in batches of 400
- Pre-caching and backup are limited to single-node OpenShift clusters only

Engineering considerations

- The **PreCachingConfig** CR is optional and does not need to be created if you only need to precache platform-related OpenShift and OLM Operator images.
- The **PreCachingConfig** CR must be applied before referencing it in the **ClusterGroupUpgrade** CR.
- Only policies with the **ran.openshift.io/ztp-deploy-wave** annotation are automatically applied by TALM during cluster installation.
- Any policy can be remediated by TALM under control of a user created **ClusterGroupUpgrade** CR.

Additional resources

- [Updating managed clusters with the Topology Aware Lifecycle Manager](#)

4.7.4. GitOps Operator and GitOps ZTP

New in this release

- No reference design updates in this release

Description

GitOps Operator and GitOps ZTP provide a GitOps-based infrastructure for managing cluster deployment and configuration. Cluster definitions and configurations are maintained as a declarative state in Git. You can apply **ClusterInstance** CRs to the hub cluster where the **SiteConfig** Operator renders them as installation CRs. In earlier releases, a GitOps ZTP plugin supported the generation of installation CRs from **SiteConfig** CRs. This plugin is now deprecated. A separate GitOps ZTP plugin is available to enable automatic wrapping of configuration CRs into policies based on the **PolicyGenerator** or **PolicyGenTemplate** CR.

You can deploy and manage multiple versions of OpenShift Container Platform on managed clusters by using the baseline reference configuration CRs. You can use custom CRs alongside the baseline CRs. To maintain multiple per-version policies simultaneously, use Git to manage the versions of the source and policy CRs by using **PolicyGenerator** or **PolicyGenTemplate** CRs.

Limits and requirements

- 300 **ClusterInstance** CRs per ArgoCD application. Multiple applications can be used to achieve the maximum number of clusters supported by a single hub cluster
- Content in the **source-crs/** directory in Git overrides content provided in the ZTP plugin container, as Git takes precedence in the search path.
- The **source-crs/** directory is specifically expected to be located in the same directory as the **kustomization.yaml** file, which includes **PolicyGenerator** or **PolicyGenTemplate** CRs as a generator. Alternative locations for the **source-crs/** directory are not supported in this context.

Engineering considerations

- For multi-node cluster upgrades, you can pause **MachineConfigPool (MCP)** CRs during maintenance windows by setting the **paused** field to **true**. You can increase the number of simultaneously updated nodes per **MCP** CR by configuring the **maxUnavailable** setting in the **MCP** CR. The **MaxUnavailable** field defines the percentage of nodes in the pool that can be simultaneously unavailable during a **MachineConfig** update. Set **maxUnavailable** to the maximum tolerable value. This reduces the number of reboots in a cluster during upgrades which results in shorter upgrade times. When you finally unpause the **MCP** CR, all the changed configurations are applied with a single reboot.
- During cluster installation, you can pause custom MCP CRs by setting the paused field to true and setting **maxUnavailable** to 100% to improve installation times.
- Keep reference CRs and custom CRs under different directories. Doing this allows you to patch and update the reference CRs by simple replacement of all directory contents without touching the custom CRs. When managing multiple versions, the following best practices are recommended:
 - Keep all source CRs and policy creation CRs in Git repositories to ensure consistent generation of policies for each OpenShift Container Platform version based solely on the contents in Git.
 - Keep reference source CRs in a separate directory from custom CRs. This facilitates easy update of reference CRs as required.
- To avoid confusion or unintentional overwrites when updating content, it is highly recommended to use unique and distinguishable names for custom CRs in the **source-crs/** directory and extra manifests in Git.
- Extra installation manifests are referenced in the **ClusterInstance** CR through a **ConfigMap** CR. The **ConfigMap** CR should be stored alongside the **ClusterInstance** CR in Git, serving as the single source of truth for the cluster. If needed, you can use a **ConfigMap** generator to create the **ConfigMap** CR.

Additional resources

- [Preparing the GitOps ZTP site configuration repository for version independence](#)
- [Adding custom content to the GitOps ZTP pipeline](#)

4.7.5. Agent-based Installer

New in this release

- No reference design updates in this release

Description

The optional Agent-based Installer component provides installation capabilities without centralized infrastructure. The installation program creates an ISO image that you mount to the server. When the server boots it installs OpenShift Container Platform and supplied extra manifests. The Agent-based Installer allows you to install OpenShift Container Platform without a hub cluster. A container image registry is required for cluster installation.

Limits and requirements

- [View the detailed list of Agent-based Installer requirements](#)

- You can supply a limited set of additional manifests at installation time.
- You must include **MachineConfiguration** CRs that are required by the RAN DU use case.

Engineering considerations

- The Agent-based Installer provides a baseline OpenShift Container Platform installation.
- You install Day 2 Operators and the remainder of the RAN DU use case configurations after installation.

Additional resources

- [Installing an OpenShift Container Platform cluster with the Agent-based Installer](#)

4.8. TELCO RAN DU REFERENCE CONFIGURATION CRs

Use the following custom resources (CRs) to configure and deploy OpenShift Container Platform clusters with the telco RAN DU profile. Use the CRs to form the common baseline used in all the specific use models unless otherwise indicated.



NOTE

You can extract the complete set of RAN DU CRs from the **ztp-site-generate** container image. See [Preparing the GitOps ZTP site configuration repository](#) for more information.

4.8.1. Cluster tuning reference CRs

Table 4.4. Cluster tuning CRs

Component	Reference CR	Description	Optional
Cluster capabilities	example-sno.yaml	Representative SiteConfig CR to install single-node OpenShift with the RAN DU profile	No
Console disable	ConsoleOperatorDisable.yaml	Disables the Console Operator.	No
Disconnected registry	09-openshift-marketplace-ns.yaml	Defines a dedicated namespace for managing the OpenShift Operator Marketplace.	No
Disconnected registry	DefaultCatsrc.yaml	Configures the catalog source for the disconnected registry.	No
Disconnected registry	DisableOLMPprof.yaml	Disables performance profiling for OLM.	No

Component	Reference CR	Description	Optional
Disconnected registry	DisconnectedICSP.yaml	Configures disconnected registry image content source policy.	No
Disconnected registry	OperatorHub.yaml	Optional, for multi-node clusters only. Configures the OperatorHub in OpenShift, disabling all default Operator sources. Not required for single-node OpenShift installs with marketplace capability disabled.	No
Monitoring configuration	ReduceMonitoringFootprint.yaml	Reduces the monitoring footprint by disabling Alertmanager and Telemeter, and sets Prometheus retention to 24 hours	No
Network diagnostics disable	DisableSnoNetworkDiag.yaml	Configures the cluster network settings to disable built-in network troubleshooting and diagnostic features.	No

4.8.2. Day 2 Operators reference CRs

Table 4.5. Day 2 Operators CRs

Component	Reference CR	Description	Optional
Cluster Logging Operator	ClusterLogForwarder.yaml	Configures log forwarding for the cluster.	No
Cluster Logging Operator	ClusterLogNS.yaml	Configures the namespace for cluster logging.	No
Cluster Logging Operator	ClusterLogOperatorGroup.yaml	Configures Operator group for cluster logging.	No

Component	Reference CR	Description	Optional
Cluster Logging Operator	ClusterLogServiceAccount.yaml	New in 4.18. Configures the cluster logging service account.	No
Cluster Logging Operator	ClusterLogServiceAccountAuditBinding.yaml	New in 4.18. Configures the cluster logging service account.	No
Cluster Logging Operator	ClusterLogServiceAccountInfrastructureBinding.yaml	New in 4.18. Configures the cluster logging service account.	No
Cluster Logging Operator	ClusterLogSubscription.yaml	Manages installation and updates for the Cluster Logging Operator.	No
Lifecycle Agent	ImageBasedUpgrade.yaml	Manage the image-based upgrade process in OpenShift.	Yes
Lifecycle Agent	LcaSubscription.yaml	Manages installation and updates for the LCA Operator.	Yes
Lifecycle Agent	LcaSubscriptionNS.yaml	Configures namespace for LCA subscription.	Yes
Lifecycle Agent	LcaSubscriptionOperatorGroup.yaml	Configures the Operator group for the LCA subscription.	Yes
Local Storage Operator	StorageClass.yaml	Defines a storage class with a Delete reclaim policy and no dynamic provisioning in the cluster.	No
Local Storage Operator	StorageLV.yaml	Configures local storage devices for the example-storage-class in the openshift-local-storage namespace, specifying device paths and filesystem type.	No

Component	Reference CR	Description	Optional
Local Storage Operator	StorageNS.yaml	Creates the namespace with annotations for workload management and the deployment wave for the Local Storage Operator.	No
Local Storage Operator	StorageOperGroup.yaml	Creates the Operator group for the Local Storage Operator.	No
Local Storage Operator	StorageSubscription.yaml	Creates the namespace for the Local Storage Operator with annotations for workload management and deployment wave.	No
LVM Operator	LVMOperatorStatus.yaml	Verifies the installation or upgrade of the LVM Storage Operator.	Yes
LVM Operator	StorageLVMCluster.yaml	Defines an LVM cluster configuration, with placeholders for storage device classes and volume group settings. Optional substitute for the Local Storage Operator.	No
LVM Operator	StorageLVMSubscription.yaml	Manages installation and updates of the LVMS Operator. Optional substitute for the Local Storage Operator.	No
LVM Operator	StorageLVMSubscriptionNS.yaml	Creates the namespace for the LVMS Operator with labels and annotations for cluster monitoring and workload management. Optional substitute for the Local Storage Operator.	No

Component	Reference CR	Description	Optional
LVM Operator	StorageLVMSubscriptionOperGroup.yaml	Defines the target namespace for the LVMS Operator. Optional substitute for the Local Storage Operator.	No
Node Tuning Operator	PerformanceProfile.yaml	Configures node performance settings in an OpenShift cluster, optimizing for low latency and real-time workloads.	No
Node Tuning Operator	TunedPerformancePatch.yaml	Applies performance tuning settings, including scheduler groups and service configurations for nodes in the specific namespace.	No
PTP fast event notifications	PtpConfigBoundaryForEvent.yaml	Configures PTP settings for PTP boundary clocks with additional options for event synchronization. Dependent on cluster role.	No
PTP fast event notifications	PtpConfigForHAForEvent.yaml	Configures PTP for highly available boundary clocks with additional PTP fast event settings. Dependent on cluster role.	No
PTP fast event notifications	PtpConfigMasterForEvent.yaml	Configures PTP for PTP grandmaster clocks with additional PTP fast event settings. Dependent on cluster role.	No
PTP fast event notifications	PtpConfigSlaveForEvent.yaml	Configures PTP for PTP ordinary clocks with additional PTP fast event settings. Dependent on cluster role.	No

Component	Reference CR	Description	Optional
PTP fast event notifications	PtpOperatorConfigForEvent.yaml	Overrides the default OperatorConfig. Configures the PTP Operator specifying node selection criteria for running PTP daemons in the openshift-ptp namespace.	No
PTP Operator	PtpConfigBoundary.yaml	Configures PTP settings for PTP boundary clocks. Dependent on cluster role.	No
PTP Operator	PtpConfigDualCardGmWpc.yaml	Configures PTP grandmaster clock settings for hosts that have dual NICs. Dependent on cluster role.	No
PTP Operator	PtpConfigThreeCardGmWpc.yaml	Configures PTP grandmaster clock settings for hosts that have 3 NICs. Dependent on cluster role.	No
PTP Operator	PtpConfigGmWpc.yaml	Configures PTP grandmaster clock settings for hosts that have a single NIC. Dependent on cluster role.	No
PTP Operator	PtpConfigSlave.yaml	Configures PTP settings for a PTP ordinary clock. Dependent on cluster role.	No
PTP Operator	PtpOperatorConfig.yaml	Configures the PTP Operator settings, specifying node selection criteria for running PTP daemons in the openshift-ptp namespace.	No

Component	Reference CR	Description	Optional
PTP Operator	PtpSubscription.yaml	Manages installation and updates of the PTP Operator in the openshift-ptp namespace.	No
PTP Operator	PtpSubscriptionNS.yaml	Configures the namespace for the PTP Operator.	No
PTP Operator	PtpSubscriptionOperatorGroup.yaml	Configures the Operator group for the PTP Operator.	No
PTP Operator (high availability)	PtpConfigBoundary.yaml	Configures PTP settings for highly available PTP boundary clocks.	No
PTP Operator (high availability)	PtpConfigForHA.yaml	Configures PTP settings for highly available PTP boundary clocks.	No
SR-IOV FEC Operator	AcceleratorsNS.yaml	Configures namespace for the VRAN Acceleration Operator. Optional part of application workload.	Yes
SR-IOV FEC Operator	AcceleratorsOperatorGroup.yaml	Configures the Operator group for the VRAN Acceleration Operator. Optional part of application workload.	Yes
SR-IOV FEC Operator	AcceleratorsSubscription.yaml	Manages installation and updates for the VRAN Acceleration Operator. Optional part of application workload.	Yes
SR-IOV FEC Operator	SriovFecClusterConfig.yaml	Configures SR-IOV FPGA Ethernet Controller (FEC) settings for nodes, specifying drivers, VF amount, and node selection.	Yes

Component	Reference CR	Description	Optional
SR-IOV Operator	SriovNetwork.yaml	Defines an SR-IOV network configuration, with placeholders for various network settings.	No
SR-IOV Operator	SriovNetworkNodePolicy.yaml	Configures SR-IOV network settings for specific nodes, including device type, RDMA support, physical function names, and the number of virtual functions.	No
SR-IOV Operator	SriovOperatorConfig.yaml	Configures SR-IOV Network Operator settings, including node selection, injector, and webhook options.	No
SR-IOV Operator	SriovOperatorConfigForSNO.yaml	Configures the SR-IOV Network Operator settings for single-node OpenShift, including node selection, injector, webhook options, and disabling node drain, in the openshift-sriov-network-operator namespace.	No
SR-IOV Operator	SriovSubscription.yaml	Manages the installation and updates of the SR-IOV Network Operator.	No
SR-IOV Operator	SriovSubscriptionNS.yaml	Creates the namespace for the SR-IOV Network Operator with specific annotations for workload management and deployment waves.	No

Component	Reference CR	Description	Optional
SR-IOV Operator	SriovSubscriptionOperatorGroup.yaml	Defines the target namespace for the SR-IOV Network Operators, enabling their management and deployment within this namespace.	No

4.8.3. Machine configuration reference CRs

Table 4.6. Machine configuration CRs

Component	Reference CR	Description	Optional
Container runtime (crun)	enable-crun-master.yaml	Configures the container runtime (crun) for control plane nodes.	No
Container runtime (crun)	enable-crun-worker.yaml	Configures the container runtime (crun) for worker nodes.	No
CRI-O wipe disable	99-crio-disable-wipe-master.yaml	Disables automatic CRI-O cache wipe following a reboot for on control plane nodes.	No
CRI-O wipe disable	99-crio-disable-wipe-worker.yaml	Disables automatic CRI-O cache wipe following a reboot for on worker nodes.	No
Kdump enable	06-kdump-master.yaml	Configures kdump crash reporting on master nodes.	No
Kdump enable	06-kdump-worker.yaml	Configures kdump crash reporting on worker nodes.	No
Kubelet configuration and container mount hiding	01-container-mount-ns-and-kubelet-conf-master.yaml	Configures a mount namespace for sharing container-specific mounts between kubelet and CRI-O on control plane nodes.	No

Component	Reference CR	Description	Optional
Kubelet configuration and container mount hiding	01-container-mount-ns-and-kubelet-conf-worker.yaml	Configures a mount namespace for sharing container-specific mounts between kubelet and CRI-O on worker nodes.	No
One-shot time sync	99-sync-time-once-master.yaml	Synchronizes time once on master nodes.	No
One-shot time sync	99-sync-time-once-worker.yaml	Synchronizes time once on worker nodes.	No
SCTP	03-sctp-machine-config-master.yaml	Loads the SCTP kernel module on master nodes.	Yes
SCTP	03-sctp-machine-config-worker.yaml	Loads the SCTP kernel module on worker nodes.	Yes
Set RCU normal	08-set rcu-normal-master.yaml	Disables rcu_expedited by setting rcu_normal after the control plane node has booted.	No
Set RCU normal	08-set rcu-normal-worker.yaml	Disables rcu_expedited by setting rcu_normal after the worker node has booted.	No
SRIOV-related kernel arguments	07-sriov-related-kernel-args-master.yaml	Enables SR-IOV support on master nodes.	No

4.9. COMPARING A CLUSTER WITH THE TELCO RAN DU REFERENCE CONFIGURATION

After you deploy a telco RAN DU cluster, you can use the **cluster-compare** plugin to assess the cluster's compliance with the telco RAN DU reference design specifications (RDS). The **cluster-compare** plugin is an OpenShift CLI (**oc**) plugin. The plugin uses a telco RAN DU reference configuration to validate the cluster with the telco RAN DU custom resources (CRs).

The plugin-specific reference configuration for telco RAN DU is packaged in a container image with the telco RAN DU CRs.

For further information about the **cluster-compare** plugin, see "Understanding the cluster-compare plugin".

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have credentials to access the **registry.redhat.io** container image registry.
- You installed the **cluster-compare** plugin.

Procedure

1. Log on to the container image registry with your credentials by running the following command:

```
$ podman login registry.redhat.io
```

2. Extract the content from the **ztp-site-generate-rhel8** container image by running the following commands::

```
$ podman pull registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.18
```

```
$ mkdir -p ./out
```

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.18 extract /home/ztp --tar | tar x -C ./out
```

3. Compare the configuration for your cluster to the reference configuration by running the following command:

```
$ oc cluster-compare -r out/reference/metadata.yaml
```

Example output

```
...
*****
Cluster CR: config.openshift.io/v1_OperatorHub_cluster ①
Reference File: required/other/operator-hub.yaml ②
Diff Output: diff -u -N /tmp/MERGED-2801470219/config-openshift-io-v1_operatorhub_cluster
/tmp/LIVE-2569768241/config-openshift-io-v1_operatorhub_cluster
--- /tmp/MERGED-2801470219/config-openshift-io-v1_operatorhub_cluster 2024-12-12
14:13:22.898756462 +0000
+++ /tmp/LIVE-2569768241/config-openshift-io-v1_operatorhub_cluster 2024-12-12
14:13:22.898756462 +0000
@@ -1,6 +1,6 @@
apiVersion: config.openshift.io/v1
kind: OperatorHub
metadata:
+ annotations: ③
+ include.release.openshift.io/hypershift: "true"
name: cluster
```

```
-spec:
- disableAllDefaultSources: true
```

Summary **4**

CRs with diffs: 11/12 **5**

CRs in reference missing from the cluster: 40 **6**

optional-image-registry:

image-registry:

Missing CRs: **7**

- optional/image-registry/ImageRegistryPV.yaml

optional-ptp-config:

ptp-config:

One of the following is required:

- optional/ptp-config/PtpConfigBoundary.yaml
- optional/ptp-config/PtpConfigGmWpc.yaml
- optional/ptp-config/PtpConfigDualCardGmWpc.yaml
- optional/ptp-config/PtpConfigForHA.yaml
- optional/ptp-config/PtpConfigMaster.yaml
- optional/ptp-config/PtpConfigSlave.yaml
- optional/ptp-config/PtpConfigSlaveForEvent.yaml
- optional/ptp-config/PtpConfigForHAForEvent.yaml
- optional/ptp-config/PtpConfigMasterForEvent.yaml
- optional/ptp-config/PtpConfigBoundaryForEvent.yaml

ptp-operator-config:

One of the following is required:

- optional/ptp-config/PtpOperatorConfig.yaml
- optional/ptp-config/PtpOperatorConfigForEvent.yaml

optional-storage:

storage:

Missing CRs:

- optional/local-storage-operator/StorageLV.yaml

...

No CRs are unmatched to reference CRs **8**

Metadata Hash:

09650c31212be9a44b99315ec14d2e7715ee194a5d68fb6d24f65fd5dbe3c3c **9**

No patched CRs **10**

- 1** The CR under comparison. The plugin displays each CR with a difference from the corresponding template.
- 2** The template matching with the CR for comparison.
- 3** The output in Linux diff format shows the difference between the template and the cluster CR.
- 4** After the plugin reports the line diffs for each CR, the summary of differences are reported.
- 5** The number of CRs in the comparison with differences from the corresponding templates.
- 6** The number of CRs represented in the reference configuration, but missing from the live cluster.

- 7** The list of CRs represented in the reference configuration, but missing from the live cluster.
- 8** The CRs that did not match to a corresponding template in the reference configuration.
- 9** The metadata hash identifies the reference configuration.
- 10** The list of patched CRs.

Additional resources

- [Understanding the cluster-compare plugin](#)

4.10. TELCO RAN DU 4.18 VALIDATED SOFTWARE COMPONENTS

The Red Hat telco RAN DU 4.18 solution has been validated using the following Red Hat software products for OpenShift Container Platform managed clusters.

Table 4.7. Telco RAN DU managed cluster validated software components

Component	Software version
Managed cluster version	4.18
Cluster Logging Operator	6.1 ¹
Local Storage Operator	4.18
OpenShift API for Data Protection (OADP)	1.4
PTP Operator	4.18
SR-IOV Operator	4.18
SRIOV-FEC Operator	2.10
Lifecycle Agent	4.18

[1] This table will be updated when the aligned Cluster Logging Operator version 6.2 is released.

4.11. TELCO RAN DU 4.18 HUB CLUSTER VALIDATED SOFTWARE COMPONENTS

The Red Hat telco RAN 4.18 solution has been validated using the following Red Hat software products for OpenShift Container Platform hub clusters.

Table 4.8. Telco hub cluster validated software components

Component	Software version
Hub cluster version	4.18
Red Hat Advanced Cluster Management (RHACM)	2.12 ¹
Red Hat OpenShift GitOps	1.14
GitOps ZTP site generate plugins	4.18
Topology Aware Lifecycle Manager (TALM)	4.18

[1] This table will be updated when the aligned RHACM version 2.13 is released.

CHAPTER 5. TELCO HUB REFERENCE DESIGN SPECIFICATIONS

The telco hub reference design specifications (RDS) describes the configuration for a hub cluster that deploys and operates fleets of OpenShift Container Platform clusters in a telco environment.



IMPORTANT

The telco hub RDS 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](#).

5.1. REFERENCE DESIGN SCOPE

The telco core, telco RAN and telco hub reference design specifications (RDS) capture the recommended, tested, and supported configurations to get reliable and repeatable performance for clusters running the telco core and telco RAN profiles.

Each RDS includes the released features and supported configurations that are engineered and validated for clusters to run the individual profiles. The configurations provide a baseline OpenShift Container Platform installation that meets feature and KPI targets. Each RDS also describes expected variations for each individual configuration. Validation of each RDS includes many long duration and at-scale tests.



NOTE

The validated reference configurations are updated for each major Y-stream release of OpenShift Container Platform. Z-stream patch releases are periodically re-tested against the reference configurations.

5.2. DEVIATIONS FROM THE REFERENCE DESIGN

Deviating from the validated telco core, telco RAN DU, and telco hub reference design specifications (RDS) can have significant impact beyond the specific component or feature that you change. Deviations require analysis and engineering in the context of the complete solution.



IMPORTANT

All deviations from the RDS should be analyzed and documented with clear action tracking information. Due diligence is expected from partners to understand how to bring deviations into line with the reference design. This might require partners to provide additional resources to engage with Red Hat to work towards enabling their use case to achieve a best in class outcome with the platform. This is critical for the supportability of the solution and ensuring alignment across Red Hat and with partners.

Deviation from the RDS can have some or all of the following consequences:

- It can take longer to resolve issues.

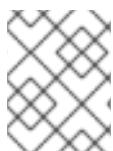
- There is a risk of missing project service-level agreements (SLAs), project deadlines, end provider performance requirements, and so on.
- Unapproved deviations may require escalation at executive levels.

**NOTE**

Red Hat prioritizes the servicing of requests for deviations based on partner engagement priorities.

5.3. HUB CLUSTER ARCHITECTURE OVERVIEW

Use the features and components running on the management hub cluster to manage many other clusters in a hub-and-spoke topology. The hub cluster provides a highly available and centralized interface for managing the configuration, lifecycle, and observability of the fleet of deployed clusters.

**NOTE**

All management hub functionality can be deployed on a dedicated OpenShift Container Platform cluster or as applications that are co-resident on an existing cluster.

Managed cluster lifecycle

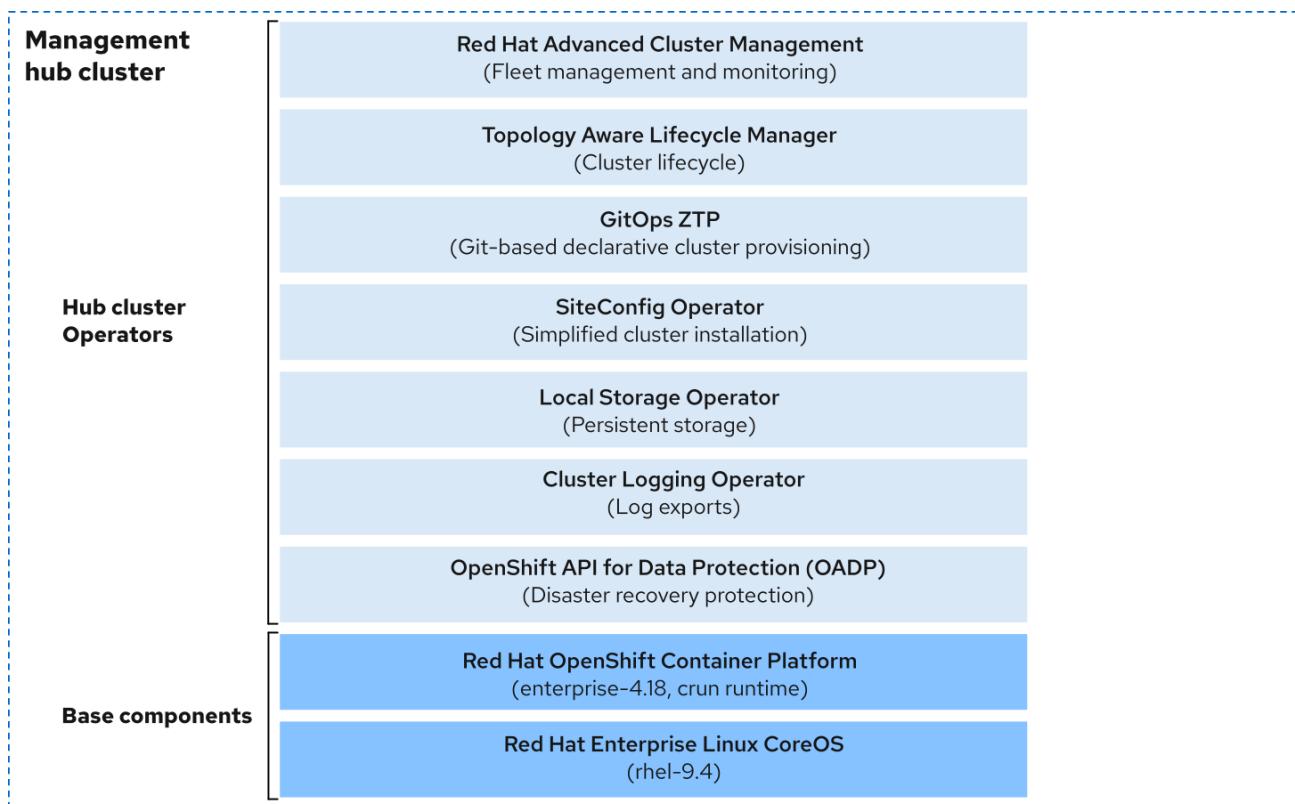
Using a combination of Day 2 Operators, the hub cluster provides the necessary infrastructure to deploy and configure the fleet of clusters by using a GitOps methodology. Over the lifetime of the deployed clusters, further management of upgrades, scaling the number of clusters, node replacement, and other lifecycle management functions can be declaratively defined and rolled out. You can control the timing and progression of the rollout across the fleet.

Monitoring

The hub cluster provides monitoring and status reporting for the managed clusters through the Observability pillar of the {rh-rhacm-full} Operator. This includes aggregated metrics, alerts, and compliance monitoring through the Governance policy framework.

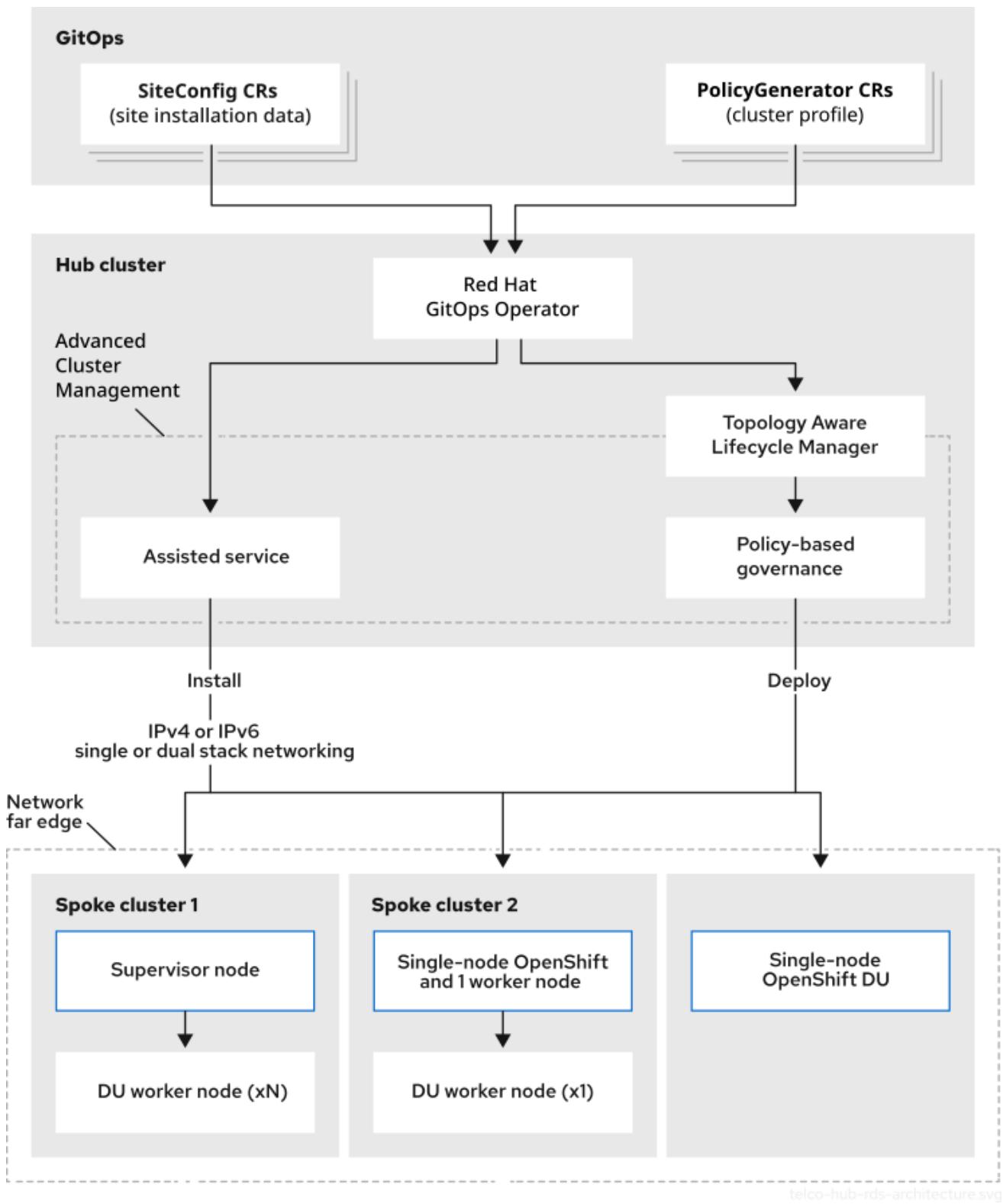
The telco management hub reference design specifications (RDS) and the associated reference custom resources (CRs) describe the telco engineering and QE validated method for deploying, configuring and managing the lifecycle of telco managed cluster infrastructure. The reference configuration includes the installation and configuration of the hub cluster components on top of OpenShift Container Platform.

Figure 5.1. Hub cluster reference design components



telco-hub-cluster-reference-design-components.svg

Figure 5.2. Hub cluster reference design architecture



5.4. TELCO MANAGEMENT HUB CLUSTER USE MODEL

The hub cluster provides managed cluster installation, configuration, observability and ongoing lifecycle management for telco application and workload clusters.

Additional resources

- For more information about core clusters or far edge clusters that host RAN distributed unit (DU) workloads, see the following:

- [Telco core RDS](#)
- [Telco RAN DU RDS](#)
- For more information about lifecycle management for the fleet of managed clusters see:
 - [Image-based upgrade for single-node OpenShift clusters](#)
 - [Updating managed clusters with the Topology Aware Lifecycle Manager](#)
 - [Upgrading a telco core CNF cluster](#)
- For more information about declarative cluster provisioning with GitOps ZTP see:
 - [Installing managed clusters with RHACM and SiteConfig resources](#)
- For more information about observability metrics and alerts, see:
 - [Multicluster architecture](#)
 - [Observability](#)

5.5. HUB CLUSTER SCALING TARGET

The resource requirements for the hub cluster are directly dependent on the number of clusters being managed by the hub, the number of policies used for each managed cluster, and the set of features that are configured in Red Hat Advanced Cluster Management (RHACM).

The hub cluster reference configuration can support up to 3500 managed single-node OpenShift clusters under the following conditions:

- 5 policies for each cluster with hub-side templating configured with a 10 minute evaluation interval.
- Only the following RHACM add-ons are enabled:
 - Policy controller
 - Observability with the default configuration
- You deploy managed clusters by using GitOps ZTP in batches of up to 500 clusters at a time.

The reference configuration is also validated for deployment and management of a mix of managed cluster topologies. The specific limits depend on the mix of cluster topologies, enabled RHACM features, and so on. In a mixed topology scenario, the reference hub configuration is validated with a combination of 1200 single-node OpenShift clusters, 400 compact clusters (3 nodes combined control plane and compute nodes), and 230 standard clusters (3 control plane and 2 worker nodes).



NOTE

Specific dimensioning requirements are highly dependent on the cluster topology and workload. For more information, see "Storage requirements". Adjust cluster dimensions for the specific characteristics of your fleet of managed clusters.

5.6. HUB CLUSTER RESOURCE UTILIZATION

Resource utilization was measured for hub clusters in the following scenario:

- Under reference load managing 3500 single-node OpenShift clusters.
- 3-node compact cluster for management hub running on dual socket bare-metal servers.
- Network impairment of 50 ms round-trip latency, 100 Mbps bandwidth limit and 0.02% packet loss.

Table 5.1. Resource utilization values

Metric	Peak Measurement
OpenShift Platform CPU	106 cores (52 cores per node)
OpenShift Platform memory	504 G (168 G per node)

Additional resources

- [Comparison of hub cluster and managed cluster templates](#)

5.7. HUB CLUSTER TOPOLOGY

In production environments, the OpenShift Container Platform hub cluster must be highly available to maintain high availability of the management functions.

Limits and requirements

Use a highly available cluster topology for the hub cluster, for example:

- Compact (3 nodes combined control plane and compute nodes)
- Standard (3 control plane nodes + N compute nodes)

Engineering considerations

- In non-production environments, a single-node OpenShift cluster can be used for limited hub cluster functionality.
- Certain capabilities, for example Red Hat OpenShift Data Foundation, are not supported on single-node OpenShift. In this configuration, some hub cluster features might not be available.
- The number of optional compute nodes can vary depending on the scale of the specific use case.
- Compute nodes can be added later as required.

Additional resources

- [OpenShift Container Platform architecture](#)
- [Postinstallation node tasks](#)

5.8. HUB CLUSTER NETWORKING

The reference hub cluster is designed to operate in a disconnected networking environment where direct access to the internet is not possible. As with all OpenShift Container Platform clusters, the hub cluster requires access to an image registry hosting all OpenShift and Day 2 Operator Lifecycle Manager (OLM) images.

The hub cluster supports dual-stack networking support for IPv6 and IPv4 networks. IPv6 is typical in edge or far-edge network segments, while IPv4 is more prevalent for use with legacy equipment in the data center.

Limits and requirements

- Regardless of the installation method, you must configure the following network types for the hub cluster:
 - **clusterNetwork**
 - **serviceNetwork**
 - **machineNetwork**
- You must configure the following IP addresses for the hub cluster:
 - **apiVIP**
 - **ingressVIP**



NOTE

For the above networking configurations, some values are required, or can be auto-assigned, depending on the chosen architecture and DHCP configuration.

- You must use the default OpenShift Container Platform network provider OVN-Kubernetes.
- Networking between the managed cluster and hub cluster must meet the networking requirements in the Red Hat Advanced Cluster Management (RHACM) documentation, for example:
 - Hub cluster access to managed cluster API service, Ironic Python agent, and baseboard management controller (BMC) port.
 - Managed cluster access to hub cluster API service, ingress IP and control plane node IP addresses.
 - Managed cluster BMC access to hub cluster control plane node IP addresses.
- An image registry must be accessible throughout the lifetime of the hub cluster.
 - All required container images must be mirrored to the disconnected registry.
 - The hub cluster must be configured to use a disconnected registry.
 - The hub cluster cannot host its own image registry. For example, the registry must be available in a scenario where a power failure affects all cluster nodes.

Engineering considerations

- When deploying a hub cluster, ensure you define appropriately sized CIDR range definitions.

Additional resources

- [Installing a cluster in a disconnected environment](#)
- [Using Operator Lifecycle Manager on restricted networks](#)
- [Configuring the hub cluster to use a disconnected mirror registry](#)
- [CIDR range definitions](#)
- [Installing OpenShift Container Platform](#)
- [Networking in OpenShift Container Platform](#)
- [Networking in RHACM](#)
- [Network configuration in RHACM](#)

5.9. HUB CLUSTER MEMORY AND CPU REQUIREMENTS

The memory and CPU requirements of the hub cluster vary depending on the configuration of the hub cluster, the number of resources on the cluster, and the number of managed clusters.

Limits and requirements

- Ensure that the hub cluster meets the underlying memory and CPU requirements for OpenShift Container Platform and Red Hat Advanced Cluster Management (RHACM).

Engineering considerations

- Before deploying a telco hub cluster, ensure that your cluster host meets cluster requirements.

For more information about scaling the number of managed clusters, see "Hub cluster scaling target".

Additional resources

- [Scaling your OpenShift Container Platform cluster and tuning performance in production environments](#)
- [Sizing your cluster](#)

5.10. HUB CLUSTER STORAGE REQUIREMENTS

The total amount of storage required by the management hub cluster is dependant on the storage requirements for each of the applications deployed on the cluster. The main components that require storage through highly available **PersistentVolume** resources are described in the following sections.

**NOTE**

The storage required for the underlying OpenShift Container Platform installation is separate to these requirements.

5.10.1. Assisted Service

The Assisted Service is deployed with the multicluster engine and Red Hat Advanced Cluster Management (RHACM).

Table 5.2. Assisted Service storage requirements

Persistent volume resource	Size (GB)
imageStorage	50
filesystemStorage	700
dataBaseStorage	20

Additional resources

- [Enabling central infrastructure management in disconnected environments](#)

5.10.2. RHACM Observability

Cluster Observability is provided by the multicluster engine and Red Hat Advanced Cluster Management (RHACM).

- Observability storage needs several **PV** resources and an S3 compatible bucket storage for long term retention of the metrics.
- Storage requirements calculation is complex and dependent on the specific workloads and characteristics of managed clusters. Requirements for **PV** resources and the S3 bucket depend on many aspects including data retention, the number of managed clusters, managed cluster workloads, and so on.
- Estimate the required storage for observability by using the observability sizing calculator in the RHACM capacity planning repository. See the Red Hat Knowledgebase article [Calculating storage need for MultiClusterHub Observability on telco environments](#) for an explanation of using the calculator to estimate observability storage requirements. The below table uses inputs derived from the telco RAN DU RDS and the hub cluster RDS as representative values.

**NOTE**

The following numbers are estimated. Tune the values for more accurate results. Add an engineering margin, for example +20%, to the results to account for potential estimation inaccuracies.

Table 5.3. Cluster requirements

Capacity planner input	Data source	Example value
Number of control plane nodes	Hub cluster RDS (scale) and telco RAN DU RDS (topology)	3500
Number of additional worker nodes	Hub cluster RDS (scale) and telco RAN DU RDS (topology)	0
Days for storage of data	Hub cluster RDS	15
Total number of pods per cluster	Telco RAN DU RDS	120
Number of namespaces (excluding OpenShift Container Platform)	Telco RAN DU RDS	4
Number of metric samples per hour	Default value	12
Number of hours of retention in receiver persistent volume (PV)	Default value	24

With these input values, the sizing calculator as described in the Red Hat Knowledgebase article [Calculating storage need for MultiClusterHub Observability on telco environments](#) indicates the following storage needs:

Table 5.4. Storage requirements

alertmanager PV		thanos receive PV		thanos compact PV	
Per replica	Total	Per replica	Total	Total	
10 GiB	30 GiB	10 GiB	30 GiB	100 GiB	

Table 5.5. Storage requirements

thanos rule PV		thanos store PV		Object bucket[1]	
Per replica	Total	Per replica	Total	Per day	Total
30 GiB	90 GiB	100 GiB	300 GiB	15 GiB	101 GiB

[1] For the object bucket, it is assumed that downsampling is disabled, so that only raw data is calculated for storage requirements.

5.10.3. Storage considerations

Limits and requirements

- Minimum OpenShift Container Platform and Red Hat Advanced Cluster Management (RHACM) limits apply
- High availability should be provided through a storage backend. The hub cluster reference configuration provides storage through Red Hat OpenShift Data Foundation.
- Object bucket storage is provided through OpenShift Data Foundation.

Engineering considerations

- Use SSD or NVMe disks with low latency and high throughput for etcd storage.
- The storage solution for telco hub clusters is OpenShift Data Foundation.
 - Local Storage Operator supports the storage class used by OpenShift Data Foundation to provide block, file, and object storage as needed by other components on the hub cluster.
- The Local Storage Operator **LocalVolume** configuration includes setting **forceWipeDevicesAndDestroyAllData: true** to support the reinstallation of hub cluster nodes where OpenShift Data Foundation has previously been used.

Additional resources

- [Persistent storage overview](#)
- [OpenShift Data Foundation architecture](#)
- [Persistent storage using local volumes](#)
- [Recommended etcd practices](#)

5.10.4. Git repository

The telco management hub cluster supports a GitOps-driven methodology for installing and managing the configuration of OpenShift clusters for various telco applications. This methodology requires an accessible Git repository that serves as the authoritative source of truth for cluster definitions and configuration artifacts.

Red Hat does not offer a commercially supported Git server. An existing Git server provided in the production environment can be used. Gitea and Gogs are examples of self-hosted Git servers that you can use.

The Git repository is typically provided in the production network external to the hub cluster. In a large-scale deployment, multiple hub clusters can use the same Git repository for maintaining the definitions of managed clusters. Using this approach, you can easily review the state of the complete network. As the source of truth for cluster definitions, the Git repository should be highly available and recoverable in disaster scenarios.



NOTE

For disaster recovery and multi-hub considerations, run the Git repository separately from the hub cluster.

Limits and requirements

- A Git repository is required to support the GitOps ZTP functions of the hub cluster, including installation, configuration, and lifecycle management of the managed clusters.
- The Git repository must be accessible from the management cluster.

Engineering considerations

- The Git repository is used by the GitOps Operator to ensure continuous deployment and a single source of truth for the applied configuration.

5.11. OPENSIFT CONTAINER PLATFORM INSTALLATION ON THE HUB CLUSTER

Description

The reference method for installing OpenShift Container Platform for the hub cluster is through the Agent-based Installer.

Agent-based Installer provides installation capabilities without additional centralized infrastructure. The Agent-based Installer creates an ISO image, which you mount to the server to be installed. When you boot the server, OpenShift Container Platform is installed alongside optionally supplied extra manifests, such as GitOps ZTP custom resources.



NOTE

You can also install OpenShift Container Platform in the hub cluster by using other installation methods.

If hub cluster functions are being applied to an existing OpenShift Container Platform cluster, the Agent-based Installer installation is not required. The remaining steps to install Day 2 Operators and configure the cluster for these functions remains the same. When OpenShift Container Platform installation is complete, the set of additional Operators and their configuration must be installed on the hub cluster.

The reference configuration includes all of these custom resources (CRs), which you can apply manually, for example:

```
$ oc apply -f <reference_cr>
```

You can also add the reference configuration to the Git repository and apply it using ArgoCD.



NOTE

If applying manually the CRs manually, take care to apply the CRs in the order indicated by the ArgoCD wave annotations. Any CRs without annotations are in the initial wave.

Limits and requirements

- Agent-based Installer requires an accessible image repository containing all required OpenShift Container Platform and Day 2 Operator images.
- Agent-based Installer builds ISO images based on a specific OpenShift releases and specific cluster details. Installation of a second hub requires a separate ISO image to be built.

Engineering considerations

- Agent-based Installer provides a baseline OpenShift Container Platform installation. You apply Day 2 Operators and other configuration CRs after the cluster is installed.
- The reference configuration supports Agent-based Installer installation in a disconnected environment.
- A limited set of additional manifests can be supplied at installation time.
- Any **MachineConfiguration** CRs you require should be included as extra manifests during installation.

Additional resources

- [OpenShift Container Platform installation overview](#)
- [Installing a cluster with customizations](#)
- [Preparing to install with the Agent-based Installer](#)

5.12. DAY 2 OPERATORS IN THE HUB CLUSTER

The management hub cluster relies on a set of Day 2 Operators to provide critical management services and infrastructure. Use Operator versions that match the set of managed cluster versions in your fleet.

Install Day 2 Operators using Operator Lifecycle Manager (OLM) and **Subscription** custom resources (CRs). **Subscription** CRs identify the specific Day 2 Operator to install, the catalog in which the Operator is found, and the appropriate version channel for the Operator. By default OLM installs and attempt to keep Operators updated with the latest z-stream version available in the channel. By default all Subscriptions are set with an **installPlanApproval: Automatic** value. In this mode, OLM automatically installs new Operator versions when they are available in the catalog and channel.



NOTE

Setting **installPlanApproval** to automatic exposes the risk of the Operator being updated outside of defined maintenance windows if the catalog index is updated to include newer Operator versions. In a disconnected environment where you are building and maintaining a curated set of Operators and versions in the catalog, and if you follow a strategy of creating a new catalog index for updated versions, the risk of the Operators being inadvertently updated is largely removed. However, if you want to further close this risk, the **Subscription** CRs can be set to **installPlanApproval: Manual** which prevents Operators from being updated without explicit administrator approval.

Limits and requirements

- When upgrading a telco hub cluster, the versions of OpenShift Container Platform and Operators must meet the requirements of all relevant compatibility matrixes.

Additional resources

- [Red Hat Advanced Cluster Management for Kubernetes 2.11 Support Matrix](#)

- [OpenShift Operator lifecycles](#)
- For more information about telco hub cluster update requirements, see:
 - [Recommended hub cluster specifications and managed cluster limits for GitOps ZTP](#)
 - [Red Hat Advanced Cluster Management for Kubernetes 2.11 Support Matrix](#)
 - [OpenShift Operator Life Cycles](#)
- For more information about updating the hub cluster, see:
 - [Introduction to OpenShift updates](#)
 - [Upgrading your hub cluster](#)
 - [Updating GitOps ZTP](#)

5.13. OBSERVABILITY

The Red Hat Advanced Cluster Management (RHACM) multicluster engine Observability component provides centralized aggregation and visualization of metrics and alerts for all managed clusters. To balance performance and data analysis, the monitoring service maintains a subset list of aggregated metrics that are collected at a downsampled interval. The metrics can be accessed on the hub through a set of different preconfigured dashboards.

Observability installation

The primary CR to enable and configure the Observability service is the **MulticlusterObservability** CR, which defines the following settings: The primary custom resource (CR) to enable and configure the observability service is the **MulticlusterObservability** CR, which defines the following settings:

- Configurable retention settings.
- Storage for the different components: **thanos receive**, **thanos compact**, **thanos rule**, **thanos store** sharding, **alertmanager**.
- The **metadata.annotations.mco-disable-alerting="true"** annotation that enables tuning for the monitoring configuration on managed clusters.



NOTE

Without this setting the Observability component attempts to configure the managed cluster monitoring configuration. With this value set you can merge your desired configuration with the necessary Observability configuration of alert forwarding into the managed cluster monitoring **ConfigMap** object. When the Observability service is enabled RHACM will deploy to each managed cluster a workload to push metrics and alerts generated by local Monitoring to the hub cluster. The metrics and alerts to be forwarded from the managed cluster to the hub, are defined by a **ConfigMap** CR in the **open-cluster-management-addon-observability** namespace. You can also specify custom metrics, for more information, see [Adding custom metrics](#).

Alertmanager configuration

- The hub cluster provides an Observability Alertmanager that can be configured to push alerts to external systems, for example, email. The Alertmanager is enabled by default.

- You must configure alert forwarding.
- When the Alertmanager is enabled but not configured, the hub Alertmanager does not forward alerts externally.
- When Observability is enabled, the managed clusters can be configured to send alerts to any endpoint including the hub Alertmanager.
- When a managed cluster is configured to forward alerts to external sources, alerts are not routed through the hub cluster Alertmanager.
- Alert state is available as a metric.
- When observability is enabled, the managed cluster alert states are included in the subset of metrics forwarded to the hub cluster and are available through Observability dashboards.

Limits and requirements

- Observability requires persistent object storage for long-term metrics. For more information, see "Storage requirements".

Engineering considerations

- Forwarding of metrics is a subset of the full metric data. It includes only the metrics defined in the **observability-metrics-allowlist** config map and any custom metrics added by the user.
- Metrics are forwarded at a downsampled rate. Metrics are forwarded by taking the latest datapoint at a 5 minute interval (or as defined by the **MultiClusterObservability** CR configuration).
- A network outage may lead to a loss of metrics forwarded to the hub cluster during that interval. This can be mitigated if metrics are also forwarded directly from managed clusters to an external metrics collector in the providers network. Full resolution metrics are available on the managed cluster.
- In addition to default metrics dashboards on the hub, users may define custom dashboards.
- The reference configuration is sized based on 15 days of metrics storage by the hub cluster for 3500 single-node OpenShift clusters. If longer retention or other managed cluster topology or sizing is required, the storage calculations must be updated and sufficient storage capacity be maintained. For more information about calculating new values, see "Storage requirements".

Additional resources

- For more information about observability, see:
 - [Exporting metrics to external endpoints](#)
 - [Enabling the Observability service](#)
- For more information about custom metrics, see [Adding custom metrics](#)
- For more information about forwarding alerts to other external systems, see [Forwarding alerts](#)

- For more information about CPU and memory requirements see: [Observability pod capacity requests](#)
- For more information about custom dashboards, see [Using Grafana dashboards](#)

5.14. MANAGED CLUSTER LIFECYCLE MANAGEMENT

To provision and manage sites at the far edge of the network, use GitOps ZTP in a hub-and-spoke architecture, where a single hub cluster manages many managed clusters.

Lifecycle management for spoke clusters can be divided into two different stages: cluster deployment, including OpenShift Container Platform installation, and cluster configuration.

Additional resources

- [Challenges of the network far edge](#)

5.14.1. Managed cluster deployment

Description

As of Red Hat Advanced Cluster Management (RHACM) 2.12, using the SiteConfig Operator is the recommended method for deploying managed clusters. The SiteConfig Operator introduces a unified ClusterInstance API that decouples the parameters that define the cluster from the manner in which it is deployed. The SiteConfig Operator uses a set of cluster templates that are instantiated using the data from a **ClusterInstance** custom resource (CR) to dynamically generate installation manifests. Following the GitOps methodology, the **ClusterInstance** CR is sourced from a Git repository through ArgoCD. The **ClusterInstance** CR can be used to initiate cluster installation by using either Assisted Installer, or the image-based installation available in multicluster engine.

Limits and requirements

- The SiteConfig ArgoCD plugin which handles **SiteConfig** CRs is deprecated from OpenShift Container Platform 4.18.

Engineering considerations

- You must create a **Secret** CR with the login information for the cluster baseboard management controller (BMC). This **Secret** CR is then referenced in the **SiteConfig** CR. Integration with a secret store, such as Vault, can be used to manage the secrets.
- Besides offering deployment method isolation and unification of Git and non-Git workflows, the SiteConfig Operator provides better scalability, greater flexibility with the use of custom templates, and an enhanced troubleshooting experience.

Additional resources

- [SiteConfig](#)
- [ClusterInstance](#)
- [Creating the managed bare-metal host secrets](#)

5.14.2. Managed cluster updates

Description

You can upgrade versions of OpenShift Container Platform, Day 2 Operators, and managed cluster configurations, by declaring the required version in the **Policy** custom resources (CRs) that target the clusters to be upgraded.

Policy controllers periodically check for policy compliance. If the result is negative, a violation report is created. If the policy remediation action is set to **enforce** the violations are remediated according to the updated policy. If the policy remediation action is set to **inform**, the process ends with a non-compliant status report and responsibility to initiate the upgrade is left to the user to perform during an appropriate maintenance window.

The Topology Aware Lifecycle Manager (TALM) extends Red Hat Advanced Cluster Management (RHACM) with features to manage the rollout of upgrades or configuration throughout the lifecycle of the fleet of clusters. It operates in progressive, limited size batches of clusters. When upgrades to OpenShift Container Platform or the Day 2 Operators are required, TALM progressively rolls out the updates by stepping through the set of policies and switching them to an "enforce" policy to push the configuration to the managed cluster.

The custom resource (CR) that TALM uses to build the remediation plan is the **ClusterGroupUpgrade** CR.

You can use image-based upgrade (IBU) with the Lifecycle Agent as an alternative upgrade path for the single-node OpenShift cluster platform version. IBU uses an OCI image generated from a dedicated seed cluster to install single-node OpenShift on the target cluster.

TALM uses the **ImageBasedGroupUpgrade** CR to roll out image-based upgrades to a set of identified clusters.

Limits and requirements

- You can perform direct upgrades for single-node OpenShift clusters using image-based upgrade for OpenShift Container Platform <4.y> to <**4.y+2**>, and <**4.y.z**> to <**4.y.z+n**>.
- Image-based upgrade uses custom images that are specific to the hardware platform that the clusters are running on. Different hardware platforms require separate seed images.

Engineering considerations

- In edge deployments, you can minimize the disruption to managed clusters by managing the timing and rollout of changes. Set all policies to **inform** to monitor compliance without triggering automatic enforcement. Similarly, configure Day 2 Operator subscriptions to manual to prevent updates from occurring outside of scheduled maintenance windows.
- The recommended upgrade approach for single-node OpenShift clusters is the image-based upgrade.
- For multi-node cluster upgrades, consider the following **MachineConfigPool** CR configurations to reduce upgrade times:
 - Pause configuration deployments to nodes during a maintenance window by setting the **paused** field to **true**.
 - Adjust the **maxUnavailable** field to control how many nodes in the pool can be updated simultaneously. The **MaxUnavailable** field defines the percentage of nodes in the pool that can be simultaneously unavailable during a **MachineConfig** object update. Set **maxUnavailable** to the maximum tolerable value. This reduces the number of reboots in a cluster during upgrades which results in shorter upgrade times.

- Resume configuration deployments by setting the **paused** field to **false**. The configuration changes are applied in a single reboot.
- During cluster installation, you can pause **MachineConfigPool** CRs by setting the **paused** field to **true** and setting **maxUnavailable** to 100% to improve installation times.

Additional resources

- [Configuration policy YAML structure](#)
- [About the ClusterGroupUpgrade CR](#)
- [Understanding the image-based upgrade for single-node OpenShift clusters](#)
- [Performing an image-based upgrade for single-node OpenShift clusters using GitOps ZTP](#)

5.15. HUB CLUSTER DISASTER RECOVERY

Note that loss of the hub cluster does not typically create a service outage on the managed clusters. Functions provided by the hub cluster will be lost, such as observability, configuration, lifecycle management updates being driven through the hub cluster, and so on.

Limits and requirements

- Backup, restore and disaster recovery are offered by the cluster backup and restore Operator, which depends on the OpenShift API for Data Protection (OADP) Operator.

Engineering considerations

- You can extend the cluster backup and restore operator to third party resources of the hub cluster based on your configuration.
- The cluster backup and restore operator is not enabled by default in Red Hat Advanced Cluster Management (RHACM). The reference configuration enables this feature.

Additional resources

- [Business continuity](#)

5.16. HUB CLUSTER COMPONENTS

5.16.1. Red Hat Advanced Cluster Management (RHACM)

New in this release

- No reference design updates in this release.

Description

Red Hat Advanced Cluster Management (RHACM) provides multicluster engine installation and ongoing lifecycle management functionality for deployed clusters. You can manage cluster configuration and upgrades declaratively by applying **Policy** custom resources (CRs) to clusters during maintenance windows.

RHACM provides functionality such as the following:

- Zero touch provisioning (ZTP) and ongoing scaling of clusters using the multicluster engine component in RHACM.
- Configuration, upgrades, and cluster status through the RHACM policy controller.
- During managed cluster installation, RHACM can apply labels to individual nodes as configured through the **ClusterInstance** CR.
- The Topology Aware Lifecycle Manager component of RHACM provides phased rollout of configuration changes to managed clusters.
- The RHACM multicluster engine Observability component provides selective monitoring, dashboards, alerts, and metrics.

The recommended method for single-node OpenShift cluster installation is the image-based installation method in multicluster engine, which uses the **ClusterInstance** CR for cluster definition.

The recommended method for single-node OpenShift upgrade is the image-based upgrade method.



NOTE

The RHACM multicluster engine Observability component brings you a centralized view of the health and status of all the managed clusters. By default, every managed cluster is enabled to send metrics and alerts, created by their Cluster Monitoring Operator (CMO), back to Observability. For more information, see "Observability".

Limits and requirements

- For more information about limits on number of clusters managed by a single hub cluster, see "Telco management hub cluster use model".
- The number of managed clusters that can be effectively managed by the hub depends on various factors, including:
 - Resource availability at each managed cluster
 - Policy complexity and cluster size
 - Network utilization
 - Workload demands and distribution
- The hub and managed clusters must maintain sufficient bi-directional connectivity.

Engineering considerations

- You can configure the cluster backup and restore Operator to include third-party resources.
- The use of RHACM hub side templating when defining configuration through policy is strongly recommended. This feature reduces the number of policies needed to manage the fleet by enabling for each cluster or for each group. For example, regional or hardware type content to be templated in a policy and substituted on cluster or group basis.

- Managed clusters typically have some number of configuration values which are specific to an individual cluster. These should be managed using RHACM policy hub side templating with values pulled from **ConfigMap** CRs based on the cluster name.

Additional resources

- [Multi Cluster Engine](#)
- [Governance](#)
- [Topology Aware Lifecycle Manager](#)
- [MultiClusterHub Observability](#)
- [Business continuity](#)
- [Performance and scalability](#)
- [Network configuration](#)

5.16.2. Topology Aware Lifecycle Manager

New in this release

- No reference design updates in this release.

Description

TALM is an Operator that runs only on the hub cluster for managing how changes like cluster upgrades, Operator upgrades, and cluster configuration are rolled out to the network. TALM supports the following features:

- Progressive rollout of policy updates to fleets of clusters in user configurable batches.
- Per-cluster actions add **ztp-done** labels or other user-configurable labels following configuration changes to managed clusters.
- TALM supports optional pre-caching of OpenShift Container Platform, OLM Operator, and additional images to single-node OpenShift clusters before initiating an upgrade. The pre-caching feature is not applicable when using the recommended image-based upgrade method for upgrading single-node OpenShift clusters.
 - Specifying optional pre-caching configurations with **PreCachingConfig** CRs.
 - Configurable image filtering to exclude unused content.
 - Storage validation before and after pre-caching, using defined space requirement parameters.

Limits and requirements

- TALM supports concurrent cluster upgrades in batches of 500.
- Pre-caching is limited to single-node OpenShift cluster topology.

Engineering considerations

- The **PreCachingConfig** custom resource (CR) is optional. You do not need to create it if you want to pre-cache platform-related images only, such as OpenShift Container Platform and OLM.
- TALM supports the use of hub-side templating with Red Hat Advanced Cluster Management policies.

5.16.3. GitOps Operator and GitOps ZTP

New in this release

- No reference design updates in this release

Description

GitOps Operator and GitOps ZTP provide a GitOps-based infrastructure for managing cluster deployment and configuration. Cluster definitions and configurations are maintained as a declarative state in Git. You can apply **ClusterInstance** custom resources (CRs) to the hub cluster where the **SiteConfig** Operator renders them as installation CRs. In earlier releases, a GitOps ZTP plugin supported the generation of installation CRs from **SiteConfig** CRs. This plugin is now deprecated. A separate GitOps ZTP plugin is available to enable automatic wrapping of configuration CRs into policies based on the **PolicyGenerator** or the **PolicyGenTemplate** CRs.

You can deploy and manage multiple versions of OpenShift Container Platform on managed clusters by using the baseline reference configuration CRs. You can use custom CRs alongside the baseline CRs. To maintain multiple per-version policies simultaneously, use Git to manage the versions of the source and policy CRs by using the **PolicyGenerator** or the **PolicyGenTemplate** CRs.

Limits and requirements

- 300 single node **SiteConfig** CRs can be synchronized for each ArgoCD application. You can use multiple applications to achieve the maximum number of clusters supported by a single hub cluster.
- To ensure consistent and complete cleanup of managed clusters and their associated resources during cluster or node deletion, you must configure ArgoCD to use background deletion mode.

Engineering considerations

- To avoid confusion or unintentional overwrite when updating content, use unique and distinguishable names for custom CRs in the **source-crs** directory and extra manifests.
- Keep reference source CRs in a separate directory from custom CRs. This facilitates easy update of reference CRs as required.
- To help with multiple versions, keep all source CRs and policy creation CRs in versioned Git repositories to ensure consistent generation of policies for each OpenShift Container Platform version.

Additional resources

- [ClusterInstance CR](#)
- [PolicyGenTemplate CRs](#)

- [GitOps ZTP version independence](#)

5.16.4. Local Storage Operator

New in this release

- No reference design updates in this release

Description

You can create persistent volumes that can be used as **PVC** resources by applications with the Local Storage Operator. The number and type of **PV** resources that you create depends on your requirements.

Engineering considerations

- Create backing storage for **PV** CRs before creating the persistent volume. This can be a partition, a local volume, LVM volume, or full disk.
- Refer to the device listing in **LocalVolume** CRs by the hardware path used to access each device to ensure correct allocation of disks and partitions, for example, **/dev/disk/by-path/<id>**. Logical names (for example, **/dev/sda**) are not guaranteed to be consistent across node reboots.

5.16.5. Red Hat OpenShift Data Foundation

New in this release

- No reference design updates in this release

Description

Red Hat OpenShift Data Foundation provides file, block, and object storage services to the hub cluster.

Limits and requirements

- Red Hat OpenShift Data Foundation (ODF) in internal mode requires the Local Storage Operator to define a storage class which will provide the necessary underlying storage.
- When doing the planning for a telco management cluster, consider the ODF infrastructure and networking requirements.
- Dual stack support is limited. ODF IPv4 is supported on dual-stack clusters.

Engineering considerations

- Address capacity warnings promptly as recovery can be difficult in case of storage capacity exhaustion, see [Capacity planning](#).

Additional resources

- [Support OpenShift dual stack with OpenShift Data Foundation using IPv4](#)
- [Infrastructure requirements](#)

- Network requirements
- Storage cluster deployment approaches

5.16.6. Logging

New in this release

- No reference design updates in this release

Description

Use the Cluster Logging Operator to collect and ship logs off the node for remote archival and analysis. The reference configuration uses Kafka to ship audit and infrastructure logs to a remote archive.

Limits and requirements

- The reference configuration does not include local log storage.
- The reference configuration does not include aggregation of managed cluster logs at the hub cluster.

Engineering considerations

- The impact of cluster CPU use is based on the number or size of logs generated and the amount of log filtering configured.
- The reference configuration does not include shipping of application logs. The inclusion of application logs in the configuration requires you to evaluate the application logging rate and have sufficient additional CPU resources allocated to the reserved set.

5.16.7. OpenShift API for Data Protection

New in this release

- No reference design updates in this release

Description

The OpenShift API for Data Protection (OADP) Operator is automatically installed and managed by Red Hat Advanced Cluster Management (RHACM) when the backup feature is enabled.

The OADP Operator facilitates the backup and restore of workloads in OpenShift Container Platform clusters. Based on the upstream open source project Velero, it allows you to backup and restore all Kubernetes resources for a given project, including persistent volumes.

While it is not mandatory to have it on the hub cluster, it is highly recommended for cluster backup, disaster recovery and high availability architecture for the hub cluster. The OADP Operator must be enabled to use the disaster recovery solutions for RHACM. The reference configuration enables backup (OADP) through the **MultiClusterHub** custom resource (CR) provided by the RHACM Operator.

Limits and requirements

- Only one version of OADP can be installed on a cluster. The version installed by RHACM must be used for RHACM disaster recovery features.

Engineering considerations

- No engineering consideration updates in this release.

5.17. HUB CLUSTER REFERENCE CONFIGURATION CRS

The following is the complete YAML reference of all the custom resources (CRs) for the telco management hub reference configuration in 4.18.

5.17.1. RHACM reference YAML

acmAgentServiceConfig.yaml

```
---
apiVersion: agent-install.openshift.io/v1beta1
kind: AgentServiceConfig
metadata:
  name: agent
  annotations:
    argocd.argoproj.io/sync-wave: "7"
    argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true
spec:
  databaseStorage:
    storageClassName: # your-fs-storageclass-here
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 20Gi
  filesystemStorage:
    storageClassName: # your-fs-storageclass-here
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 20Gi
  imageStorage:
    storageClassName: # your-fs-storageclass-here
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 100Gi
  mirrorRegistryRef:
    name: mirror-registry-config
  osImages:
    # Replace <http-server-address:port> with the address of the local web server that stores the RHCOS images.
    # The images can be downloaded from "https://mirror.openshift.com/pub/openshift-v4/x86_64/dependencies/rhcos/".
    - cpuArchitecture: "x86_64"
      openshiftVersion: "4.17"
```

```
rootFSUrl: http://<http-server-address:port>/rhcos-4.17.0-x86_64-live-rootfs.x86_64.img
url: http://<http-server-address:port>/rhcos-4.17.0-x86_64-live.x86_64.iso
version: "417.94.202409121747-0"
```

acmMCH.yaml

```
---
apiVersion: operator.open-cluster-management.io/v1
kind: MultiClusterHub
metadata:
  annotations:
    argocd.argoproj.io/sync-wave: "4"
    argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true
    installer.open-cluster-management.io/mce-subscription-spec: '{"source": "redhat-operators-disconnected", "installPlanApproval": "Automatic"}'
    installer.open-cluster-management.io/oadp-subscription-spec: '{"source": "redhat-operators-disconnected", "installPlanApproval": "Automatic"}'
  name: multiclusetherub
  namespace: open-cluster-management
spec:
  availabilityConfig: High
  enableClusterBackup: false
  ingress: {}
  overrides:
    components:
      - configOverrides: {}
        enabled: true
        name: app-lifecycle
      - configOverrides: {}
        enabled: true
        name: cluster-lifecycle
      - configOverrides: {}
        enabled: true
        name: cluster-permission
      - configOverrides: {}
        enabled: true
        name: console
      - configOverrides: {}
        enabled: true
        name: grc
      - configOverrides: {}
        enabled: true
        name: insights
      - configOverrides: {}
        enabled: true
        name: multiclusether-engine
      - configOverrides: {}
        enabled: true
        name: multiclusether-observability
      - configOverrides: {}
        enabled: true
        name: search
      - configOverrides: {}
        enabled: true
        name: submariner-addon
      - configOverrides: {}
```

```

enabled: true
name: volsync
- configOverrides: {}
  enabled: true
  name: cluster-backup
- configOverrides: {}
  enabled: true
  name: siteconfig
- configOverrides: {}
  enabled: false
  name: edge-manager-preview
separateCertificateManagement: false

```

acmMirrorRegistryCM.yaml

```

---
apiVersion: v1
kind: ConfigMap
metadata:
  name: mirror-registry-config
  annotations:
    argocd.argoproj.io/sync-wave: "5"
    argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true
  namespace: multicluster-engine
  labels:
    app: assisted-service
data:
  # Add the mirror registry SSL certificate chain up to the CA itself.
  ca-bundle.crt: |
    -----BEGIN CERTIFICATE-----
    MIID7jCCAtagAwXXX...
    -----END CERTIFICATE-----
    -----BEGIN CERTIFICATE-----
    MIIDvTCCAqWgAwXXX...
    -----END CERTIFICATE-----
  # The registries.conf field has been populated using the registries.conf file found in
  #"/etc/containers/registries.conf" on each node.
  # Replace <registry.example.com:8443> with the mirror registry's address.
  registries.conf: |
    unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]
    [[registry]]
    prefix = ""
    location = "quay.io/openshift-release-dev"

    [[registry.mirror]]
    location = "<registry.example.com:8443>/openshift-release-dev"
    pull-from-mirror = "digest-only"

    [[registry]]
    prefix = ""
    location = "quay.io/openshift-release-dev/ocp-release"

    [[registry.mirror]]
    location = "<registry.example.com:8443>/openshift-release-dev/ocp-release"
    pull-from-mirror = "digest-only"

```

```
[[registry]]
prefix = ""
location = "quay.io/openshift-release-dev/ocp-v4.0-art-dev"

[[registry.mirror]]
location = "<registry.example.com:8443>/openshift-release-dev/ocp-v4.0-art-dev"
pull-from-mirror = "digest-only"

[[registry]]
prefix = ""
location = "registry.redhat.io/multicluster-engine"

[[registry.mirror]]
location = "<registry.example.com:8443>/multicluster-engine"
pull-from-mirror = "digest-only"

[[registry]]
prefix = ""
location = "registry.redhat.io/odf4"

[[registry.mirror]]
location = "<registry.example.com:8443>/odf4"
pull-from-mirror = "digest-only"

[[registry]]
prefix = ""
location = "registry.redhat.io/openshift4"

[[registry.mirror]]
location = "<registry.example.com:8443>/openshift4"
pull-from-mirror = "digest-only"

[[registry]]
prefix = ""
location = "registry.redhat.io/rhacm2"

[[registry.mirror]]
location = "<registry.example.com:8443>/rhacm2"
pull-from-mirror = "digest-only"

[[registry]]
prefix = ""
location = "registry.redhat.io/rhceph"

[[registry.mirror]]
location = "<registry.example.com:8443>/rhceph"
pull-from-mirror = "digest-only"

[[registry]]
prefix = ""
location = "registry.redhat.io/rhel8"

[[registry.mirror]]
location = "<registry.example.com:8443>/rhel8"
pull-from-mirror = "digest-only"
```

```
[[registry]]
prefix = ""
location = "registry.redhat.io/rhel9"

[[registry.mirror]]
location = "<registry.example.com:8443>/rhel9"
pull-from-mirror = "digest-only"

[[registry]]
prefix = ""
location = "registry.redhat.io/ubi8"

[[registry.mirror]]
location = "<registry.example.com:8443>/ubi8"
pull-from-mirror = "tag-only"
```

acmNS.yaml

```
---
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
  name: open-cluster-management
```

acmOperGroup.yaml

```
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: open-cluster-management-group
  namespace: open-cluster-management
spec:
  targetNamespaces:
  - open-cluster-management
```

acmPerfSearch.yaml

```
---
apiVersion: search.open-cluster-management.io/v1alpha1
kind: Search
metadata:
  name: search-v2-operator
  namespace: open-cluster-management
  annotations:
    argocd.argoproj.io/sync-wave: "10"
    argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true
spec:
  dbStorage:
    size: 10Gi
  deployments:
    collector:
```

```

resources:
limits:
  memory: 8Gi
requests:
  cpu: 25m
  memory: 64Mi
database:
envVar:
- name: POSTGRESQL_EFFECTIVE_CACHE_SIZE
  value: 1024MB
- name: POSTGRESQL_SHARED_BUFFERS
  value: 512MB
- name: WORK_MEM
  value: 128MB
resources:
limits:
  memory: 16Gi
requests:
  cpu: 25m
  memory: 32Mi
indexer:
resources:
limits:
  memory: 4Gi
requests:
  cpu: 25m
  memory: 128Mi
queryapi:
replicaCount: 2
resources:
limits:
  memory: 4Gi
requests:
  cpu: 25m
  memory: 1Gi
tolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/infra
  operator: Exists

```

acmProvisioning.yaml

```

---
apiVersion: metal3.io/v1alpha1
kind: Provisioning
metadata:
  name: provisioning-configuration
annotations:
  argocd.argoproj.io/sync-wave: "6"
  argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true
spec:
  watchAllNamespaces: true
  # some servers do not support virtual media installations
  # when the image is served using the https protocol
  # disableVirtualMediaTLS: true

```

acmSubscription.yaml

```
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: open-cluster-management-subscription
  namespace: open-cluster-management
spec:
  channel: release-2.13
  installPlanApproval: Automatic
  name: advanced-cluster-management
  source: redhat-operators-disconnected
  sourceNamespace: openshift-marketplace
```

observabilityMCO.yaml

```
---
apiVersion: observability.open-cluster-management.io/v1beta2
kind: MultiClusterObservability
metadata:
  name: observability
  annotations:
    argocd.argoproj.io/sync-wave: "10"
    argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true
    # avoids MultiClusterHub Observability to own/manage the
    # spoke clusters configuration about AlertManager forwards.
    # ZTP Policies will be in charge of configuring it
    # https://issues.redhat.com/browse/CNF-13398
    mco-disable-alerting: "true"
spec:
  # based on the data provided by acm-capacity tool
  # https://github.com/stolostron/capacity-planning/blob/main/calculation/ObsSizingTemplate-Rev1.ipynb
  # for an scenario with:
  # 3500SNOs, 125 pods and 4 Namespaces (apart from Openshift NS)
  # storage retention 15 days
  # downsampling disabled
  # default MCO Addon configuration samples_per_hour, pv_retention_hrs.
  # More on how to estimate: https://access.redhat.com/articles/7103886
  advanced:
    retentionConfig:
      blockDuration: 2h
      deleteDelay: 48h
      retentionInLocal: 24h
      retentionResolutionRaw: 15d
    enableDownsampling: false
    observabilityAddonSpec:
      enableMetrics: true
      interval: 300
    storageConfig:
      storageClass: # your-fs-storageclass-here
      alertmanagerStorageSize: 10Gi
      compactStorageSize: 100Gi
      metricObjectStorage:
```

```

key: thanos.yaml
name: thanos-object-storage
receiveStorageSize: 10Gi
ruleStorageSize: 30Gi
storeStorageSize: 100Gi
# In addition to these storage settings, the `metricObjectStorage`
# points to an Object Storage. Under the reference configuration,
# scale and retention the estimated object storage is about 101Gi

```

observabilityNS.yaml

```

---
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
  name: open-cluster-management-observability

```

observabilityOBC.yaml

```

---
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: observability-obc
  annotations:
    argocd.argoproj.io/sync-wave: "8"
    argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true
  namespace: open-cluster-management-observability
spec:
  generateBucketName: observability-object-bucket
  storageClassName: openshift-storage.noobaa.io

```

observabilitySecret.yaml

```

---
apiVersion: v1
kind: Secret
metadata:
  annotations:
    argocd.argoproj.io/sync-wave: "9"
    argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true
  labels:
    cluster.open-cluster-management.io/backup: ""
  name: multiclusetherhub-operator-pull-secret
  namespace: open-cluster-management-observability
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: " # Value provided by user or by pull-secret-openshift-config-copy policy"

```

thanosSecret.yaml

```

# This content creates a policy which copies the necessary data from

```

```

# the generated Object Bucket Claim into the necessary secret for
# observability to connect to thanos.

---
apiVersion: v1
kind: Namespace
metadata:
  name: hub-policies
---

apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  annotations:
    policy.open-cluster-management.io/categories: CM Configuration Management
    policy.open-cluster-management.io/controls: CM-2 Baseline Configuration
    policy.open-cluster-management.io/description: ""
    policy.open-cluster-management.io/standards: NIST SP 800-53
    argocd.argoproj.io/sync-wave: "9"
    argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true
  name: obs-thanos-secret
  namespace: hub-policies
spec:
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: thanos-secret-cp
        spec:
          remediationAction: enforce
          severity: high
          object-templates-raw: |
            {{- /* read the bucket data and noobaa endpoint access data */ -}}
            {{- $objBucket := (lookup "v1" "ConfigMap" "open-cluster-management-observability"
"observability-obc") --}}
            {{- $awsAccess := (lookup "v1" "Secret" "open-cluster-management-observability"
"observability-obc") --}}
            {{- /* create the thanos config file as a template */ -}}
            {{- $thanosConfig := `}}
              type: s3
              config:
                bucket: %[1]s
                endpoint: %[2]s
                insecure: true
                access_key: %[3]s
                secret_key: %[4]s
            `}}
            {{- /* create the secret using the thanos configuration template created above. */ -}}
            - complianceType: mustonlyhave
              objectDefinition:
                apiVersion: v1
                kind: Secret
                metadata:
                  name: thanos-object-storage
                  namespace: open-cluster-management-observability

```

```

type: Opaque
data:
  thanos.yaml: {{ (printf $thanوسConfig $objBucket.data.BUCKET_NAME
    $objBucket.data.BUCKET_HOST
    ($awsAccess.data.AWS_ACCESS_KEY_ID | base64dec)
    ($awsAccess.data.AWS_SECRET_ACCESS_KEY | base64dec)
  ) | base64enc }}

---
apiVersion: cluster.open-cluster-management.io/v1beta1
kind: Placement
metadata:
  name: obs-thanos-pl
  namespace: hub-policies
  annotations:
    argocd.argoproj.io/sync-wave: "9"
    argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true
spec:
  predicates:
    - requiredClusterSelector:
        labelSelector:
          matchExpressions:
            - key: name
              operator: In
              values:
                - local-cluster
---
apiVersion: policy.open-cluster-management.io/v1
kind: PlacementBinding
metadata:
  name: obs-thanos-binding
  namespace: hub-policies
  annotations:
    argocd.argoproj.io/sync-wave: "9"
    argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true
placementRef:
  name: obs-thanos-pl
  apiGroup: cluster.open-cluster-management.io
  kind: Placement
subjects:
  - name: obs-thanos-secret
    apiGroup: policy.open-cluster-management.io
    kind: Policy
---
apiVersion: cluster.open-cluster-management.io/v1beta2
kind: ManagedClusterSetBinding
metadata:
  name: default
  namespace: hub-policies
  annotations:
    argocd.argoproj.io/sync-wave: "8"
    argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true
spec:
  clusterSet: default

# For reference this is the secret which is being generated (with
# appropriate values in the fields):

```

```

# ---
# apiVersion: v1
# kind: Secret
# metadata:
#   name: thanos-object-storage
#   namespace: open-cluster-management-observability
#   type: Opaque
#   stringData:
#     thanos.yaml: |
#       type: s3
#       config:
#         bucket: "<BUCKET_NAME>"
#         endpoint: "<BUCKET_HOST>"
#         insecure: true
#         access_key: "<AWS_ACCESS_KEY_ID>"
#         secret_key: "<AWS_SECRET_ACCESS_KEY>"
```

talmSubscription.yaml

```

---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-topology-aware-lifecycle-manager-subscription
  namespace: openshift-operators
spec:
  channel: stable
  installPlanApproval: Automatic
  name: topology-aware-lifecycle-manager
  source: redhat-operators-disconnected
  sourceNamespace: openshift-marketplace
```

5.17.2. Storage reference YAML**IsoLocalVolume.yaml**

```

---
apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
  name: "local-disks"
  namespace: "openshift-local-storage"
  annotations:
    argocd.argoproj.io/sync-wave: "2"
    argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true
spec:
  nodeSelector:
    nodeSelectorTerms:
      - matchExpressions:
          - key: cluster.ocs.openshift.io/openshift-storage
            operator: In
            values:
              - ""
storageClassDevices:
```

```

- storageClassName: "local-sc"
  forceWipeDevicesAndDestroyAllData: true
  volumeMode: Block
  devicePaths:
    - /dev/disk/by-path/pci-xxx

```

IsoNS.yaml

```

---
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-local-storage
  labels:
    openshift.io/cluster-monitoring: "true"

```

IsoOperatorgroup.yaml

```

---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: local-operator-group
  namespace: openshift-local-storage
spec:
  targetNamespaces:
    - openshift-local-storage

```

IsoSubscription.yaml

```

---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: local-storage-operator
  namespace: openshift-local-storage
spec:
  channel: stable
  installPlanApproval: Automatic
  name: local-storage-operator
  source: redhat-operators-disconnected
  sourceNamespace: openshift-marketplace

```

odfNS.yaml

```

---
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-storage
  annotations:
    workload.openshift.io/allowed: management
  labels:
    openshift.io/cluster-monitoring: "true"

```

odfOperatorGroup.yaml

```
---  
apiVersion: operators.coreos.com/v1  
kind: OperatorGroup  
metadata:  
  name: openshift-storage-operatorgroup  
  namespace: openshift-storage  
spec:  
  targetNamespaces:  
    - openshift-storage
```

odfSubscription.yaml

```
---  
apiVersion: operators.coreos.com/v1alpha1  
kind: Subscription  
metadata:  
  name: odf-operator  
  namespace: openshift-storage  
spec:  
  channel: "stable-4.18"  
  name: odf-operator  
  source: redhat-operators-disconnected  
  sourceNamespace: openshift-marketplace  
  installPlanApproval: Automatic
```

storageCluster.yaml

```
---  
apiVersion: ocs.openshift.io/v1  
kind: StorageCluster  
metadata:  
  name: ocs-storagecluster  
  namespace: openshift-storage  
  annotations:  
    argocd.argoproj.io/sync-wave: "3"  
    argocd.argoproj.io/sync-options: SkipDryRunOnMissingResource=true  
spec:  
  manageNodes: false  
  resources:  
    mds:  
      limits:  
        cpu: "3"  
        memory: "8Gi"  
      requests:  
        cpu: "3"  
        memory: "8Gi"  
  monDataDirHostPath: /var/lib/rook  
  storageDeviceSets:  
    - count: 1 #-- Modify count to desired value. For each set of 3 disks increment the count by 1.  
    dataPVCTemplate:  
      spec:
```

```

accessModes:
- ReadWriteOnce
resources:
requests:
storage: "600Gi" # <-- This should be changed as per storage size. Minimum 100 GiB and
Maximum 4 TiB
storageClassName: "local-sc" # match this with the storage block created at the LSO step
volumeMode: Block
name: ocs-deviceset
placement: {}
portable: false
replica: 3
resources:
limits:
cpu: "2"
memory: "5Gi"
requests:
cpu: "2"
memory: "5Gi"

```

5.17.3. GitOps Operator and GitOps ZTP reference YAML

`argocd-ssh-known-hosts-cm.yaml`

```

---
apiVersion: v1
kind: ConfigMap
metadata:
  name: argocd-ssh-known-hosts-cm
  namespace: openshift-gitops
data:
  ssh_known_hosts: |
#####
# by default empty known hosts, because of usual      #
# disconnected environments.                      #
#                                                 #
# Manually add needed ssh known hosts:          #
# example: $> ssh-keyscan my-github.com          #
# Copy the output here
#####
# my-github.com sh-rsa AAAAB3NzaC1y...J4i36KV/aCl4Ixz
# my-github.com ecdsa-sha2-nistp256...GGtLKqmwLLeKhe6xgc=
# my-github.com ssh-ed25519 AAAAC3N...INrvWjBQ2u

```

`gitopsNS.yaml`

```

---
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-gitops-operator
  labels:
    openshift.io/cluster-monitoring: "true"

```

gitopsOperatorGroup.yaml

```
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-gitops-operator
  namespace: openshift-gitops-operator
spec:
  upgradeStrategy: Default
```

gitopsSubscription.yaml

```
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-gitops-operator
  namespace: openshift-gitops-operator
spec:
  channel: gitops-1.15
  installPlanApproval: Automatic
  name: openshift-gitops-operator
  source: redhat-operators-disconnected
  sourceNamespace: openshift-marketplace
```

ztp-repo.yaml

```
---
apiVersion: v1
kind: Secret
metadata:
  name: ztp-repo
  namespace: openshift-gitops
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  # use following for ssh repo access
  url: git@gitlab.example.com:namespace/repo.git
  insecure: "false"
  sshPrivateKey: |
    -----BEGIN OPENSSH PRIVATE KEY-----
    INSERT PRIVATE KEY
    -----END OPENSSH PRIVATE KEY-----
  # uncomment and use following for https repo access
  # url: https://gitlab.example.com/namespace/repo
  # insecure: "false"
  # password: password
  # username: username
  # forceHttpBasicAuth: "true"
  # more examples: https://argo-cd.readthedocs.io/en/stable/operator-manual/argocd-repositories-yaml/
```

app-project.yaml

```

apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: ztp-app-project
  namespace: openshift-gitops
  annotations:
    argocd.argoproj.io/sync-wave: "100"
spec:
  clusterResourceWhitelist:
    - group: 'hive.openshift.io'
      kind: ClusterImageSet
    - group: 'cluster.open-cluster-management.io'
      kind: ManagedCluster
    - group: ""
      kind: Namespace
  destinations:
    - namespace: "*"
      server: '*'
  namespaceResourceWhitelist:
    - group: ""
      kind: ConfigMap
    - group: ""
      kind: Namespace
    - group: ""
      kind: Secret
    - group: 'agent-install.openshift.io'
      kind: InfraEnv
    - group: 'agent-install.openshift.io'
      kind: NMStateConfig
    - group: 'extensions.hive.openshift.io'
      kind: AgentClusterInstall
    - group: 'extensions.hive.openshift.io'
      kind: ImageClusterInstall
    - group: 'hive.openshift.io'
      kind: ClusterDeployment
    - group: 'metal3.io'
      kind: BareMetalHost
    - group: 'metal3.io'
      kind: HostFirmwareSettings
    - group: 'metal3.io'
      kind: DataImage
    - group: 'agent.open-cluster-management.io'
      kind: KlusterletAddonConfig
    - group: 'cluster.open-cluster-management.io'
      kind: ManagedCluster
    - group: 'ran.openshift.io'
      kind: SiteConfig
    - group: 'siteconfig.open-cluster-management.io'
      kind: ClusterInstance
  sourceRepos:
    - "*"

```

argocd Openshift Gitops Patch

```
{

```

```
"spec": {
  "controller": {
    "resources": {
      "limits": {
        "cpu": "16",
        "memory": "32Gi"
      },
      "requests": {
        "cpu": "1",
        "memory": "2Gi"
      }
    }
  },
  "kustomizeBuildOptions": "--enable-alpha-plugins",
  "repo": {
    "volumes": [
      {
        "name": "kustomize",
        "emptyDir": {}
      }
    ],
    "initContainers": [
      {
        "resources": {},
        "terminationMessagePath": "/dev/termination-log",
        "name": "kustomize-plugin",
        "command": [
          "/exportkustomize.sh"
        ],
        "args": [
          "./config"
        ],
        "imagePullPolicy": "Always",
        "volumeMounts": [
          {
            "name": "kustomize",
            "mountPath": "./config"
          }
        ],
        "terminationMessagePolicy": "File",
        "image": "registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.17.0"
      },
      {
        "args": [
          "-C",
          "mkdir -p ./config/kustomize/plugin/policy.open-cluster-management.io/v1/policygenerator &&
cp /policy-generator/PolicyGenerator-not-fips-compliant ./config/kustomize/plugin/policy.open-cluster-
management.io/v1/policygenerator/PolicyGenerator"
        ],
        "command": [
          "/bin/bash"
        ],
        "image": "registry.redhat.io/rhacm2/multioperator-subscription-rhel9:v2.11",
        "name": "policy-generator-install",
        "imagePullPolicy": "Always",
      }
    ]
  }
}
```

```

"volumeMounts": [
  {
    "mountPath": "./config",
    "name": "kustomize"
  }
],
"volumeMounts": [
  {
    "name": "kustomize",
    "mountPath": "./config"
  }
],
"env": [
  {
    "name": "ARGOCD_EXEC_TIMEOUT",
    "value": "360s"
  },
  {
    "name": "KUSTOMIZE_PLUGIN_HOME",
    "value": "./config/kustomize/plugin"
  }
],
"resources": {
  "limits": {
    "cpu": "8",
    "memory": "16Gi"
  },
  "requests": {
    "cpu": "1",
    "memory": "2Gi"
  }
}
}
}
}

```

clusters-app.yaml

```

apiVersion: v1
kind: Namespace
metadata:
  name: clusters-sub
  annotations:
    argocd.argoproj.io/sync-wave: "100"
---
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: clusters
  namespace: openshift-gitops
  annotations:
    argocd.argoproj.io/sync-wave: "100"
spec:
  destination:

```

```

server: https://kubernetes.default.svc
namespace: clusters-sub
project: ztp-app-project
source:
  path: ztp/gitops-subscriptions/argocd/example/siteconfig
  repoURL: https://github.com/openshift-kni/cnf-features-deploy
  targetRevision: master
  # uncomment the below plugin if you will be adding the plugin binaries in the same repo->dir where
  # the sitconfig.yaml exist AND use the ../../hack/patch-argocd-dev.sh script to re-patch the
  deployment-repo-server
# plugin:
#   name: kustomize-with-local-plugins
ignoreDifferences: # recommended way to allow ACM controller to manage its fields. alternative
approach documented below (1)
- group: cluster.open-cluster-management.io
  kind: ManagedCluster
  managedFieldsManagers:
    - controller
# (1) alternatively you can choose to ignore a specific path like so (replace managedFieldsManagers
with jsonPointers)
#   jsonPointers:
#     - /metadata/labels/cloud
#     - /metadata/labels/vendor
syncPolicy:
  automated:
    prune: true
    selfHeal: true
syncOptions:
  - CreateNamespace=true
  - PrunePropagationPolicy=background
  - RespectIgnoreDifferences=true

```

gitops-cluster-rolebinding.yaml

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: gitops-cluster
  annotations:
    argocd.argoproj.io/sync-wave: "100"
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: openshift-gitops-argocd-application-controller
    namespace: openshift-gitops

```

gitops-policy-rolebinding.yaml

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: gitops-policy

```

```

annotations:
  argocd.argoproj.io/sync-wave: "100"
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: open-cluster-management:cluster-manager-admin
subjects:
- kind: ServiceAccount
  name: openshift-gitops-argocd-application-controller
  namespace: openshift-gitops

```

kustomization.yaml

```

apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- app-project.yaml
- policies-app-project.yaml
- gitops-policy-rolebinding.yaml
- gitops-cluster-rolebinding.yaml
- clusters-app.yaml
- policies-app.yaml
- AddPluginsPolicy.yaml

```

policies-app-project.yaml

```

---
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: policy-app-project
  namespace: openshift-gitops
annotations:
  argocd.argoproj.io/sync-wave: "100"
spec:
  clusterResourceWhitelist:
  - group: ""
    kind: Namespace
  destinations:
  - namespace: 'ztp*'
    server: '*'
  - namespace: 'policies-sub'
    server: '*'
  namespaceResourceWhitelist:
  - group: ""
    kind: ConfigMap
  - group: ""
    kind: Namespace
  - group: 'apps.open-cluster-management.io'
    kind: PlacementRule
  - group: 'policy.open-cluster-management.io'
    kind: Policy
  - group: 'policy.open-cluster-management.io'
    kind: PlacementBinding

```

```

- group: 'ran.openshift.io'
  kind: PolicyGenTemplate
- group: cluster.open-cluster-management.io
  kind: Placement
- group: policy.open-cluster-management.io
  kind: PolicyGenerator
- group: policy.open-cluster-management.io
  kind: PolicySet
- group: cluster.open-cluster-management.io
  kind: ManagedClusterSetBinding
sourceRepos:
- '*'

```

policies-app.yaml

```

apiVersion: v1
kind: Namespace
metadata:
  name: policies-sub
  annotations:
    argocd.argoproj.io/sync-wave: "100"
---
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: policies
  namespace: openshift-gitops
  annotations:
    argocd.argoproj.io/sync-wave: "100"
spec:
  destination:
    server: https://kubernetes.default.svc
    namespace: policies-sub
    project: policy-app-project
  source:
    path: ztp/gitops-subscriptions/argocd/example/policygentemplates
    repoURL: https://github.com/openshift-kni/cnf-features-deploy
    targetRevision: master
    # uncomment the below plugin if you will be adding the plugin binaries in the same repo->dir where
    # the policyGenTemplate.yaml exist AND use the ../../hack/patch-argocd-dev.sh script to re-patch
    # the deployment-repo-server
    # plugin:
    #   name: kustomize-with-local-plugins
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true

```

5.17.4. Logging reference YAML

clusterLogNS.yaml

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-logging
  annotations:
    workload.openshift.io/allowed: management
```

clusterLogOpenGroup.yaml

```
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: cluster-logging
  namespace: openshift-logging
spec:
  targetNamespaces:
    - openshift-logging
```

clusterLogSubscription.yaml

```
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: cluster-logging
  namespace: openshift-logging
spec:
  channel: "stable-6.2"
  name: cluster-logging
  source: redhat-operators-disconnected
  sourceNamespace: openshift-marketplace
  installPlanApproval: Automatic
```

5.17.5. Installation reference YAML

agent-config.yaml

```
---
apiVersion: v1beta1
kind: AgentConfig
metadata:
  name: hub # need to match the same name put in install-config
  rendezvousIP: 192.168.125.20 # one of the master IP
  # Replace the fields below with your network details
hosts:
  - hostname: hub-ctl-0
    role: master
    interfaces:
      - name: ens3
        macAddress: aa:aa:aa:aa:01:01
    networkConfig:
```

```
interfaces:
  - name: ens3
    mac-address: aa:aa:aa:aa:01:01
  ipv4:
    enabled: true
    dhcp: true
  ipv6:
    enabled: true
    dhcp: false
    address:
      - ip: fd01::20
        prefix-length: 64
routes:
  config:
    - destination: ::/0
      next-hop-address: fd01::1
      next-hop-interface: ens3
      table-id: 254
rootDeviceHints:
  deviceName: "/dev/disk/by-path/pci-0000:00:07.0"
- hostname: hub-ctl-1
  role: master
  interfaces:
    - name: ens3
      macAddress: aa:aa:aa:aa:01:02
networkConfig:
  interfaces:
    - name: ens3
      mac-address: aa:aa:aa:aa:01:02
      ipv4:
        enabled: true
        dhcp: true
      ipv6:
        enabled: true
        dhcp: false
        address:
          - ip: fd01::21
            prefix-length: 64
      routes:
        config:
          - destination: ::/0
            next-hop-address: fd01::1
            next-hop-interface: ens3
            table-id: 254
  rootDeviceHints:
    deviceName: "/dev/disk/by-path/pci-0000:00:07.0"
- hostname: hub-ctl-2
  role: master
  interfaces:
    - name: ens3
      macAddress: aa:aa:aa:aa:01:03
networkConfig:
  interfaces:
    - name: ens3
      mac-address: aa:aa:aa:aa:01:03
      ipv4:
```

```

enabled: true
dhcp: true
ipv6:
  enabled: true
  dhcp: false
  address:
    - ip: fd01::22
      prefix-length: 64
routes:
  config:
    - destination: ::/0
      next-hop-address: fd01::1
      next-hop-interface: ens3
      table-id: 254
rootDeviceHints:
  deviceName: "/dev/disk/by-path/pci-0000:00:07.0"

```

install-config.yaml

```

---
apiVersion: v1
metadata:
  name: hub # replace with your hub name
baseDomain: example.com # replace with your domain name
compute:
- architecture: amd64
  hyperthreading: Enabled
  name: worker
  replicas: 0
controlPlane:
  architecture: amd64
  hyperthreading: Enabled
  name: master
  replicas: 3
networking:
  clusterNetwork:
    - cidr: 10.128.0.0/14
      hostPrefix: 23
    - cidr: fd02::/48
      hostPrefix: 64
  machineNetwork:
    - cidr: 192.168.125.0/24 # replace with your machine network CIDR
    - cidr: fd01::/64
  networkType: OVNKubernetes
  serviceNetwork:
    - 172.30.0.0/16
    - fd03::/112
# Replace the fields below with your network details
platform:
  baremetal:
    provisioningNetwork: "Disabled"
    apiVIPs:
      - 192.168.125.10
      - fd01::10
    ingressVIPs:
      - 192.168.125.11

```

```

- fd01::11
# Replace <registry.example.com:8443> with the mirror registry's address.
imageDigestSources:
- mirrors:
  - <registry.example.com:8443>/openshift-release-dev/ocp-release
    source: quay.io/openshift-release-dev/ocp-release
- mirrors:
  - <registry.example.com:8443>/openshift-release-dev/ocp-v4.0-art-dev
    source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
# Add the mirror registry SSL certificate chain up to the CA itself.
additionalTrustBundle: |
  -----BEGIN CERTIFICATE-----
  MIID7jCCAtagAwXXX...
  -----END CERTIFICATE-----
  -----BEGIN CERTIFICATE-----
  MIIDvTCCAqWgAwIBAgIUcXQpXXX...
  -----END CERTIFICATE-----
# Add the mirror registry credentials to the pull secret.
pullSecret: '{"auths": {"<registry.example.com:8443>": {"auth": "aW5pdDo0R1XXXXXjdCbUoweUNuMWI1OTZBMmhkcEhjMw==", "email": "user@redhat.com"}, ...}}}'
# Add the SSH public key to connect to the OCP nodes
sshKey: |
  ssh-rsa AAAAB3NzaC1yc2EA...

```

5.18. TELCO HUB REFERENCE CONFIGURATION SOFTWARE SPECIFICATIONS

The telco hub 4.18 solution has been validated using the following Red Hat software products for OpenShift Container Platform clusters.

Table 5.6. Telco hub cluster validated software components

Component	Software version
OpenShift Container Platform	4.18
Local Storage Operator	4.18
Red Hat OpenShift Data Foundation (ODF)	4.18
Red Hat Advanced Cluster Management (RHACM)	2.13
Red Hat OpenShift GitOps	1.15
GitOps Zero Touch Provisioning (ZTP) plugins	4.18
multicluster engine Operator PolicyGenerator plugin	2.12
Topology Aware Lifecycle Manager (TALM)	4.18
Cluster Logging Operator	6.2

Component	Software version
OpenShift API for Data Protection (OADP)	The version aligned with the RHACM release.

CHAPTER 6. COMPARING CLUSTER CONFIGURATIONS

6.1. UNDERSTANDING THE CLUSTER-COMPARE PLUGIN

The **cluster-compare** plugin is an OpenShift CLI (**oc**) plugin that compares a cluster configuration with a reference configuration. The plugin reports configuration differences while suppressing expected variations by using configurable validation rules and templates.

Use the **cluster-compare** plugin in development, production, and support scenarios to ensure cluster compliance with a reference configuration, and to quickly identify and troubleshoot relevant configuration differences.

6.1.1. Overview of the cluster-compare plugin

Clusters deployed at scale typically use a validated set of baseline custom resources (CRs) to configure clusters to meet use-case requirements and ensure consistency when deploying across different environments.

In live clusters, some variation from the validated set of CRs is expected. For example, configurations might differ because of variable substitution, optional components, or hardware-specific fields. This variation makes it difficult to accurately assess if a cluster is compliant with the baseline configuration.

Using the **cluster-compare** plugin with the **oc** command, you can compare the configuration from a live cluster with a reference configuration. A reference configuration represents the baseline configuration but uses the various plugin features to suppresses expected variation during a comparison. For example, you can apply validation rules, specify optional and required resources, and define relationships between resources. By reducing irrelevant differences, the plugin makes it easier to assess cluster compliance with baseline configurations, and across environments.

The ability to intelligently compare a configuration from a cluster with a reference configuration has the following example use-cases:

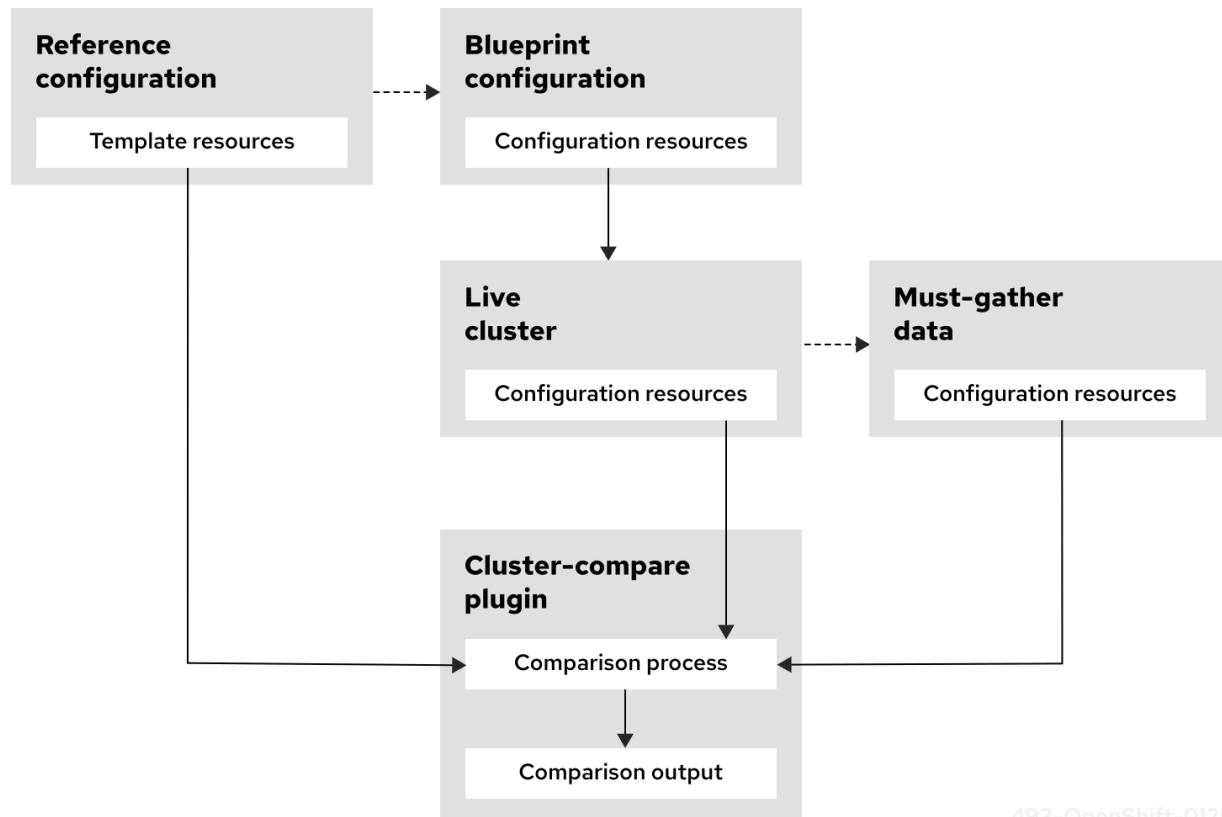
Production: Ensure compliance with a reference configuration across service updates, upgrades and changes to the reference configuration.

Development: Ensure compliance with a reference configuration in test pipelines.

Design: Compare configurations with a partner lab reference configuration to ensure consistency.

Support: Compare the reference configuration to **must-gather** data from a live cluster to troubleshoot configuration issues.

Figure 6.1. Cluster-compare plugin overview

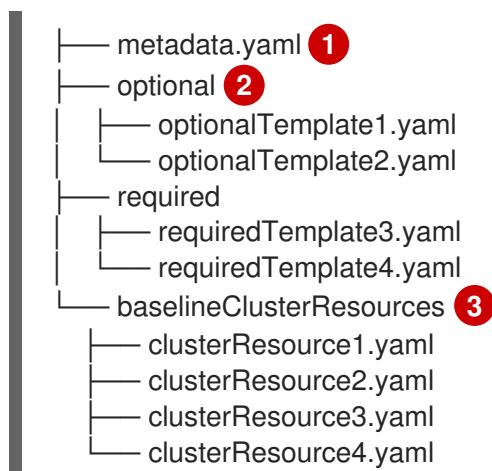


493-OpenShift-0125

6.1.2. Understanding a reference configuration

The **cluster-compare** plugin uses a reference configuration to validate a configuration from a live cluster. The reference configuration consists of a YAML file called **metadata.yaml**, which references a set of templates that represent the baseline configuration.

Example directory structure for a reference configuration



- ① The reference configuration consists of the **metadata.yaml** file and a set of templates.
- ② This example uses an optional and required directory structure for templates that are referenced by the **metadata.yaml** file.
- ③ The configuration CRs to use as a baseline configuration for clusters.

During a comparison, the plugin matches each template to a configuration resource from the cluster. The plugin evaluates optional or required fields in the template using features such as Golang templating syntax and inline regular expression validation. The **metadata.yaml** file applies additional validation rules to decide whether a template is optional or required and assesses template dependency relationships.

Using these features, the plugin identifies relevant configuration differences between the cluster and the reference configuration. For example, the plugin can highlight mismatched field values, missing resources, extra resources, field type mismatches, or version discrepancies.

For further information about configuring a reference configuration, see "Creating a reference configuration".

6.1.3. Additional resources

- [Telco RAN DU reference design specification for OpenShift Container Platform](#)

6.2. INSTALLING THE CLUSTER-COMPARE PLUGIN

You can extract the **cluster-compare** plugin from a container image in the Red Hat container catalog and use it as a plugin to the **oc** command.

6.2.1. Installing the cluster-compare plugin

Install the **cluster-compare** plugin to compare a reference configuration with a cluster configuration from a live cluster or **must-gather** data.

Prerequisites

1. You have installed the OpenShift CLI (**oc**).
2. You installed **podman**.
3. You have access to the Red Hat container catalog.

Procedure

1. Log in to the Red Hat container catalog by running the following command:

```
$ podman login registry.redhat.io
```

2. Create a container for the **cluster-compare** image by running the following command:

```
$ podman create --name cca registry.redhat.io/openshift4/kube-compare-artifacts-rhel9:latest
```

3. Copy the **cluster-compare** plugin to a directory that is included in your **PATH** environment variable by running the following command:

```
$ podman cp cca:/usr/share/openshift/<arch>/kube-compare.<rhel_version><directory_on_path>/kubectl-cluster_compare
```

- **arch** is the architecture for your machine. Valid values are:

- **linux_amd64**

- **linux_arm64**
- **linux_ppc64le**
- **linux_s390x**
- **<rhel_version>** is the version of RHEL on your machine. Valid values are **rhel8** or **rhel9**.
- **<directory_on_path>** is the path to a directory included in your **PATH** environment variable.

Verification

- View the help for the plugin by running the following command:

```
$ oc cluster-compare -h
```

Example output

Compare a known valid reference configuration and a set of specific cluster configuration CRs.

...

Usage:

```
compare -r <Reference File>
```

Examples:

```
# Compare a known valid reference configuration with a live cluster:  
kubectl cluster-compare -r ./reference/metadata.yaml
```

...

6.2.2. Additional resources

- [Extending the OpenShift CLI with plugins](#)

6.3. USING THE CLUSTER-COMPARE PLUGIN

You can use the **cluster-compare** plugin to compare a reference configuration with a configuration from a live cluster or **must-gather** data.

6.3.1. Using the cluster-compare plugin with a live cluster

You can use the **cluster-compare** plugin to compare a reference configuration with configuration custom resources (CRs) from a live cluster.

Validate live cluster configurations to ensure compliance with reference configurations during design, development, or testing scenarios.



NOTE

Use the **cluster-compare** plugin with live clusters in non-production environments only.
For production environments, use the plugin with **must-gather** data.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You downloaded the **cluster-compare** plugin and include it in your **PATH** environment variable.
- You have access to a reference configuration.

Procedure

- Run the **cluster-compare** plugin by using the following command:

```
$ oc cluster-compare -r <path_to_reference_config>/metadata.yaml
```

- **-r** specifies a path to the **metadata.yaml** file of the reference configuration. You can specify a local directory or a URI.

Example output

```
...
*****
Cluster CR: operator.openshift.io/v1_Console_cluster ①
Reference File: optional/console-disable/ConsoleOperatorDisable.yaml ②
Diff Output: diff -u -N /tmp/MERGED-622469311/operator-openshift-io-v1_console_cluster /tmp/LIVE-2358803347/operator-openshift-io-v1_console_cluster /tmp/MERGED-622469311/operator-openshift-io-v1_console_cluster 2024-11-20
15:43:42.888633602 +0000
+++ /tmp/LIVE-2358803347/operator-openshift-io-v1_console_cluster 2024-11-20
15:43:42.888633602 +0000
@@ -4,5 +4,5 @@
    name: cluster
  spec:
    logLevel: Normal
- managementState: Removed ③
+ managementState: Managed
    operatorLogLevel: Normal
*****
...
Summary ④
CRs with diffs: 5/49 ⑤
CRs in reference missing from the cluster: 1 ⑥
required-cluster-tuning:
  cluster-tuning:
```

Missing CRs: 7

- required/cluster-tuning/disabling-network-diagnostics/DisableSnoNetworkDiag.yaml

No CRs are unmatched to reference CRs 8

Metadata Hash:

512a9bf2e57fd5a5c44bbdea7abb3ffd7739d4a1f14ef9021f6793d5cdf868f0 9

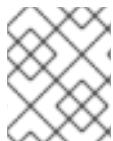
No patched CRs 10

- 1 The CR under comparison. The plugin displays each CR with a difference from the corresponding template.
- 2 The template matching with the CR for comparison.
- 3 The output in Linux diff format shows the difference between the template and the cluster CR.
- 4 After the plugin reports the line diffs for each CR, the summary of differences are reported.
- 5 The number of CRs in the comparison with differences from the corresponding templates.
- 6 The number of CRs represented in the reference configuration, but missing from the live cluster.
- 7 The list of CRs represented in the reference configuration, but missing from the live cluster.
- 8 The CRs that did not match to a corresponding template in the reference configuration.
- 9 The metadata hash identifies the reference configuration.
- 10 The list of patched CRs.

6.3.2. Using the cluster-compare plugin with must-gather data

You can use the **cluster-compare** plugin to compare a reference configuration with configuration custom resources (CRs) from **must-gather** data.

Validate cluster configurations by using **must-gather** data to troubleshoot configuration issues in production environments.



NOTE

For production environments, use the **cluster-compare** plugin with **must-gather** data only.

- You have access to **must-gather** data from a target cluster.
- You installed the OpenShift CLI (**oc**).
- You have downloaded the **cluster-compare** plugin and included it in your **PATH** environment variable.

- You have access to a reference configuration.

Procedure

- Compare the **must-gather** data to a reference configuration by running the following command:

```
$ oc cluster-compare -r <path_to_reference_config>/metadata.yaml -f "must-gather/*/cluster-scoped-resources","must-gather/*/namespaces" -R
```

- **-r** specifies a path to the **metadata.yaml** file of the reference configuration. You can specify a local directory or a URI.
- **-f** specifies the path to the **must-gather** data directory. You can specify a local directory or a URI. This example restricts the comparison to the relevant cluster configuration directories.
- **-R** searches the target directories recursively.

Example output

```
...
*****
Cluster CR: operator.openshift.io/v1_Console_cluster ①
Reference File: optional/console-disable/ConsoleOperatorDisable.yaml ②
Diff Output: diff -u -N /tmp/MERGED-622469311/operator-openshift-io-
v1_console_cluster /tmp/LIVE-2358803347/operator-openshift-io-v1_console_cluster
/tmp/MERGED-622469311/operator-openshift-io-v1_console_cluster 2024-11-20
15:43:42.888633602 +0000
+++ /tmp/LIVE-2358803347/operator-openshift-io-v1_console_cluster 2024-11-20
15:43:42.888633602 +0000
@@ -4,5 +4,5 @@
    name: cluster
  spec:
    logLevel: Normal
- managementState: Removed ③
+ managementState: Managed
    operatorLogLevel: Normal
*****
...
Summary ④
CRs with diffs: 5/49 ⑤
CRs in reference missing from the cluster: 1 ⑥
required-cluster-tuning:
  cluster-tuning:
    Missing CRs: ⑦
      - required/cluster-tuning/disabling-network-diagnostics/DisableSnoNetworkDiag.yaml
No CRs are unmatched to reference CRs ⑧
```

Metadata Hash:

512a9bf2e57fd5a5c44bbdea7abb3ffd7739d4a1f14ef9021f6793d5cdf868f0 9

No patched CRs 10

- 1 The CR under comparison. The plugin displays each CR with a difference from the corresponding template.
- 2 The template matching with the CR for comparison.
- 3 The output in Linux diff format shows the difference between the template and the cluster CR.
- 4 After the plugin reports the line diffs for each CR, the summary of differences are reported.
- 5 The number of CRs in the comparison with differences from the corresponding templates.
- 6 The number of CRs represented in the reference configuration, but missing from the live cluster.
- 7 The list of CRs represented in the reference configuration, but missing from the live cluster.
- 8 The CRs that did not match to a corresponding template in the reference configuration.
- 9 The metadata hash identifies the reference configuration.
- 10 The list of patched CRs.

Additional resources

- [Gathering data about your cluster](#)

6.3.3. Reference cluster-compare plugin options

The following content describes the options for the **cluster-compare** plugin.

Table 6.1. Cluster-compare plugin options

Option	Description
-A, --all-resources	When used with a live cluster, attempts to match all resources in the cluster that match a type in the reference configuration. When used with local files, attempts to match all resources in the local files that match a type in the reference configuration.
--concurrency	Specify an integer value for the number of templates to process in parallel when comparing with resources from the live version. A larger number increases speed but also memory, I/O, and CPU usage during that period. The default value is 4 .

Option	Description
-c, --diff-config	Specify the path to the user configuration file.
-f, --filename	Specify a filename, directory, or URL for the configuration custom resources that you want to use for a comparison with a reference configuration.
--generate-override-for	Specify the path for templates that requires a patch.  NOTE You must use a file path for the target template that is relative to the metadata.yaml file. For example, if the file path for the metadata.yaml file is ./compare/metadata.yaml , a relative file path for the template might be optional/my-template.yaml .
-h, --help	Display help information.
-k, --kustomize	Specify a path to process the kustomization directory. This flag cannot be used together with -f or -R .
-o, --output	Specify the output format. Options include json , yaml , or generate-patches .
--override-reason	Specify a reason for generating the override.
-p, --overrides	Specify a path to a patch override file for the reference configuration.
-R, --recursive	Processes the directory specified in -f, --filename recursively.
-r, --reference	Specify the path to the reference configuration metadata.yaml file.
--show-managed-fields	Specify true to include managed fields in the comparison.
-v, --verbose	Increases the verbosity of the plugin output.

6.3.4. Example: Comparing a cluster with the telco core reference configuration

You can use the **cluster-compare** plugin to compare a reference configuration with a configuration from a live cluster or **must-gather** data.

This example compares a configuration from a live cluster with the telco core reference configuration. The telco core reference configuration is derived from the telco core reference design specifications (RDS). The telco core RDS is designed for clusters to support large scale telco applications including control plane and some centralized data plane functions.

The reference configuration is packaged in a container image with the telco core RDS.

For further examples of using the **cluster-compare** plugin with the telco core and telco RAN distributed unit (DU) profiles, see the "Additional resources" section.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have credentials to access the **registry.redhat.io** container image registry.
- You installed the **cluster-compare** plugin.

Procedure

1. Log on to the container image registry with your credentials by running the following command:

```
$ podman login registry.redhat.io
```

2. Extract the content from the **telco-core-rds-rhel9** container image by running the following commands:

```
$ mkdir -p ./out
```

```
$ podman run -it registry.redhat.io/openshift4/openshift-telco-core-rds-rhel9:v4.18 | base64 -d  
| tar xv -C out
```

You can view the reference configuration in the **reference-crs-kube-compare/** directory.

```
out/telco-core-rds/configuration/reference-crs-kube-compare/  
└── metadata.yaml ①  
    └── optional ②  
        ├── logging  
        ├── networking  
        ├── other  
        └── tuning  
    └── required ③  
        ├── networking  
        ├── other  
        ├── performance  
        ├── scheduling  
        └── storage
```

- 1 Configuration file for the reference configuration.
- 2 Directory for optional templates.
- 3 Directory for required templates.

3. Compare the configuration for your cluster to the telco core reference configuration by running the following command:

```
$ oc cluster-compare -r out/telco-core-rds/configuration/reference-crs-kube-
compare/metadata.yaml
```

Example output

```
W1212 14:13:06.281590 36629 compare.go:425] Reference Contains Templates With
Types (kind) Not Supported By Cluster: BFDProfile, BGPAAdvertisement, BGPPeer,
ClusterLogForwarder, Community, IPAddressPool, MetaLB, MultiNetworkPolicy, NMState,
NUMAResourcesOperator, NUMAResourcesScheduler, NodeNetworkConfigurationPolicy,
SriovNetwork, SriovNetworkNodePolicy, SriovOperatorConfig, StorageCluster
```

...

```
*****
```

Cluster CR: config.openshift.io/v1_OperatorHub_cluster ①

Reference File: required/other/operator-hub.yaml ②

Diff Output: diff -u -N /tmp/MERGED-2801470219/config-openshift-io-v1_operatorhub_cluster
/tmp/LIVE-2569768241/config-openshift-io-v1_operatorhub_cluster

```
--- /tmp/MERGED-2801470219/config-openshift-io-v1_operatorhub_cluster 2024-12-12
14:13:22.898756462 +0000
```

```
+++ /tmp/LIVE-2569768241/config-openshift-io-v1_operatorhub_cluster 2024-12-12
14:13:22.898756462 +0000
```

@@ -1,6 +1,6 @@

apiVersion: config.openshift.io/v1

kind: OperatorHub

metadata:

- + annotations: ③

- + include.release.openshift.io/hypershift: "true"

- name: cluster

- spec:

- disableAllDefaultSources: true

```
*****
```

Summary ④

CRs with diffs: 3/4 ⑤

CRs in reference missing from the cluster: 22 ⑥

other:

other:

 Missing CRs: ⑦

- optional/other/control-plane-load-kernel-modules.yaml
- optional/other/worker-load-kernel-modules.yaml

required-networking:

 networking-root:

 Missing CRs:

- required/networking/nodeNetworkConfigurationPolicy.yaml

 networking-sriov:

 Missing CRs:

- required/networking/sriov/sriovNetwork.yaml
- required/networking/sriov/sriovNetworkNodePolicy.yaml
- required/networking/sriov/SriovOperatorConfig.yaml
- required/networking/sriov/SriovSubscription.yaml
- required/networking/sriov/SriovSubscriptionNS.yaml

```

- required/networking/sriov/SriovSubscriptionOperGroup.yaml
required-other:
  scheduling:
    Missing CRs:
      - required/other/catalog-source.yaml
      - required/other/icsp.yaml
required-performance:
  performance:
    Missing CRs:
      - required/performance/PerformanceProfile.yaml
required-scheduling:
  scheduling:
    Missing CRs:
      - required/scheduling/nrop.yaml
      - required/scheduling/NROPSSubscription.yaml
      - required/scheduling/NROPSSubscriptionNS.yaml
      - required/scheduling/NROPSSubscriptionOperGroup.yaml
      - required/scheduling/sched.yaml
required-storage:
  storage-odf:
    Missing CRs:
      - required/storage/odf-external/01-rook-ceph-external-cluster-details.secret.yaml
      - required/storage/odf-external/02-ocs-external-storagecluster.yaml
      - required/storage/odf-external/odfNS.yaml
      - required/storage/odf-external/odfOperGroup.yaml
      - required/storage/odf-external/odfSubscription.yaml
No CRs are unmatched to reference CRs ⑧
Metadata Hash:
fe41066bac56517be02053d436c815661c9fa35eec5922af25a1be359818f297 ⑨
No patched CRs ⑩

```

- ① The CR under comparison. The plugin displays each CR with a difference from the corresponding template.
- ② The template matching with the CR for comparison.
- ③ The output in Linux diff format shows the difference between the template and the cluster CR.
- ④ After the plugin reports the line diffs for each CR, the summary of differences are reported.
- ⑤ The number of CRs in the comparison with differences from the corresponding templates.
- ⑥ The number of CRs represented in the reference configuration, but missing from the live cluster.
- ⑦ The list of CRs represented in the reference configuration, but missing from the live cluster.
- ⑧ The CRs that did not match to a corresponding template in the reference configuration.
- ⑨ The metadata hash identifies the reference configuration.
- ⑩ The list of patched CRs.

6.3.5. Additional resources

- Comparing a cluster with the telco RAN DU reference configuration
- Comparing a cluster with the telco core reference configuration

6.4. CREATING A REFERENCE CONFIGURATION

Configure a reference configuration to validate configuration resources from a cluster.

6.4.1. Structure of the metadata.yaml file

The **metadata.yaml** file provides a central configuration point to define and configure the templates in a reference configuration. The file features a hierarchy of **parts** and **components**. **parts** are groups of **components** and **components** are groups of templates. Under each component, you can configure template dependencies, validation rules, and add descriptive metadata.

Example metadata.yaml file

```
apiVersion: v2
parts: ①
  - name: Part1 ②
    components:
      - name: Component1 ③
        <component1_configuration> ④
      - name: Part2
        - name: Component2
          <component2_configuration>
```

- ① Every **part** typically describes a workload or a set of workloads.
- ② Specify a **part** name.
- ③ Specify a **component** name.
- ④ Specify the configuration for a template. For example, define template relationships or configure what fields to use in a comparison.

6.4.2. Configuring template relationships

By defining relationships between templates in your reference configuration, you can support use-cases with complex dependencies. For example, you can configure a component to require specific templates, require one template from a group, or allow any template from a group, and so on.

Procedure

- Create a **metadata.yaml** file to match your use case. Use the following structure as an example:

Example metadata.yaml file

```
apiVersion: v2
parts:
  - name: Part1
```

```

components:
  - name: Component1
    allOf: ①
      - path: RequiredTemplate1.yaml
      - path: RequiredTemplate2.yaml
  - name: Component2
    allOrNoneOf: ②
      - path: OptionalBlockTemplate1.yaml
      - path: OptionalBlockTemplate2.yaml
  - name: Component3
    anyOf: ③
      - path: OptionalTemplate1.yaml
      - path: OptionalTemplate2.yaml
  - name: Component4
    noneOf: ④
      - path: BannedTemplate1.yaml
      - path: BannedTemplate2.yaml
  - name: Component5
    oneOf: ⑤
      - path: RequiredExclusiveTemplate1.yaml
      - path: RequiredExclusiveTemplate2.yaml
  - name: Component6
    anyOneOf: ⑥
      - path: OptionalExclusiveTemplate1.yaml
      - path: OptionalExclusiveTemplate2.yaml
#...

```

- ① Specifies required templates.
- ② Specifies a group of templates that are either all required or all optional. If one corresponding custom resource (CR) is present in the cluster, then all corresponding CRs must be present in the cluster.
- ③ Specifies optional templates.
- ④ Specifies templates to exclude. If a corresponding CR is present in the cluster, the plugin returns a validation error.
- ⑤ Specifies templates where only one can be present. If none, or more than one of the corresponding CRs are present in the cluster, the plugin returns a validation error .
- ⑥ Specifies templates where only one can be present in the cluster. If more than one of the corresponding CRs are present in the cluster, the plugin returns a validation error.

6.4.3. Configuring expected variation in a template

You can handle variable content within a template by using Golang templating syntax. Using this syntax, you can configure validation logic that handles optional, required, and conditional content within the template.



NOTE

- The **cluster-compare** plugin requires all templates to render as valid YAML. To avoid parsing errors for missing fields, use conditional templating syntax such as `{{- if .spec.<optional_field> }}` when implementing templating syntax. This conditional logic ensures templates process missing fields gracefully and maintains valid YAML formatting.
- You can use the Golang templating syntax with custom and built-in functions for complex use cases. All Golang built-in functions are supported including the functions in the Sprig library.

Procedure

- Create a **metadata.yaml** file to match your use case. Use the following structure as an example:

```
apiVersion: v2
kind: Service
metadata:
  name: frontend ①
  namespace: {{ .metadata.namespace }} ②
  labels:
    app: guestbook
    tier: frontend
spec:
  {{- if and .spec.type (eq (.spec.type) "NodePort" "LoadBalancer") }}
  type: {{.spec.type }} ③
  {{- else }}
  type: should be NodePort or LoadBalancer
  {{- end }}
  ports:
  - port: 80
  selector:
    app: guestbook
    {{- if .spec.selector.tier }} ④
    tier: frontend
    {{- end }}
```

- ① Configures a required field that must match the specified value.
- ② Configures a required field that can have any value.
- ③ Configures validation for the **.spec.type** field.
- ④ Configures an optional field.

6.4.4. Configuring the metadata.yaml file to exclude template fields

You can configure the **metadata.yaml** file to exclude fields from a comparison. Exclude fields that are irrelevant to a comparison, for example annotations or labels that are inconsequential to a cluster configuration.

You can configure exclusions in the **metadata.yaml** file in the following ways:

- Exclude all fields in a custom resource not specified in a template.
- Exclude specific fields that you define using the **pathToKey** field.

**NOTE**

pathToKey is a dot separated path. Use quotes to escape key values featuring a period.

6.4.4.1. Excluding all fields not specified in a template

During the comparison process, the **cluster-compare** plugin renders a template by merging fields from the corresponding custom resource (CR). If you configure the **ignore-unspecified-fields** to **true**, all fields that are present in the CR, but not in the template, are excluded from the merge. Use this approach when you want to focus the comparison on the fields specified in the template only.

Procedure

- Create a **metadata.yaml** file to match your use case. Use the following structure as an example:

```
apiVersion: v2
parts:
- name: Part1
components:
- name: Namespace
  allOf:
  - path: namespace.yaml
    config:
      ignore-unspecified-fields: true ①
#...
```

- Specify **true** to exclude from the comparison all fields in a CR that are not explicitly configured in the corresponding **namespace.yaml** template.

6.4.4.2. Excluding specific fields by setting default exclusion fields

You can exclude fields by defining a default value for **fieldsToOmitRefs** in the **defaultOmitRef** field. This default exclusion applies to all templates, unless overridden by the **config.fieldsToOmitRefs** field for a specific template.

Procedure

- Create a **metadata.yaml** file to match your use case. Use the following structure as an example:

Example metadata.yaml file

```
apiVersion: v2
parts:
#...
fieldsToOmit:
  defaultOmitRef: default ①
```

```

items:
  default:
    - pathToKey: a.custom.default."k8s.io" ②

```

- ① Sets the default exclusion for all templates, unless overridden by the **config.fieldsToOmitRefs** field for a specific template.
- ② The value is excluded for all templates.

6.4.4.3. Excluding specific fields

You can specify fields to exclude by defining the path to the field, and then referencing the definition in the **config** section for a template.

Procedure

- Create a **metadata.yaml** file to match your use case. Use the following structure as an example:

Example metadata.yaml file

```

apiVersion: v2
parts:
  - name: Part1
components:
  - name: Component1
    - path: deployment.yaml
config:
  fieldsToOmitRefs:
    - deployments ①
#...
fieldsToOmit:
  items:
    deployments:
      - pathToKey: spec.selector.matchLabels.k8s-app ②

```

- ① References the **fieldsToOmit.items.deployments** item for the **deployment.yaml** template.
- ② Excludes the **spec.selector.matchLabels.k8s-app** field from the comparison.



NOTE

Setting **fieldsToOmitRefs** replaces the default value.

6.4.4.4. Excluding specific fields by setting default exclusion groups

You can create default groups of fields to exclude. A group of exclusions can reference another group to avoid duplication when defining exclusions.

Procedure

- Create a **metadata.yaml** file to match your use case. Use the following structure as an example:

Example metadata.yaml file

```
apiVersion: v2
parts:
#...

fieldsToOmit:
defaultOmitRef: default
items:
common:
- pathToKey: metadata.annotations."kubernetes.io/metadata.name"
- pathToKey: metadata.annotations."kubernetes.io/metadata.name"
- pathToKey: metadata.annotations."kubectl.kubernetes.io/last-applied-configuration"
- pathToKey: metadata.creationTimestamp
- pathToKey: metadata.generation
- pathToKey: spec.ownerReferences
- pathToKey: metadata.ownerReferences
default:
- include: common ①
- pathToKey: status
```

- ① The **common** group is included in the default group.

6.4.5. Configuring inline validation for template fields

You can enable inline regular expressions to validate template fields, especially in scenarios where Golang templating syntax is difficult to maintain or overly complex. Using inline regular expressions simplifies templates, improves readability, and allows for more advanced validation logic.

The **cluster-compare** plugin provides two functions for inline validation:

- **regex**: Validates content in a field using a regular expression.
- **capturegroups**: Enhances multi-line text comparisons by processing non-capture group text as exact matches, applying regular expression matching only within named capture groups, and ensuring consistency for repeated capture groups.

6.4.5.1. Configuring inline validation with the regex function

Use the **regex** inline function to validate fields using regular expressions.

Procedure

1. Create a **metadata.yaml** file to match your use case. Use the following structure as an example:

```
apiVersion: v2
parts:
- name: Part1
components:
- name: Example
allOf:
```

```

- path: example.yaml
  config:
    perField:
      - pathToKey: spec.bigTextBlock ①
        inlineDiffFunc: regex ②
  
```

- ① Specifies the field for inline validation.
- ② Enables inline validation using regular expressions.

2. Use a regular expression to validate the field in the associated template:

```

apiVersion: v1
kind: ConfigMap
metadata:
  namespace: dashboard
data:
  bigTextBlock: |-  

    This is a big text block with some static content, like this line.  

    It also has a place where (?<username>[a-z0-9]+) would put in their own name. (?<username>[a-z0-9]+) would put in their own name.
  
```

6.4.5.2. Configuring inline validation with the `capturegroups` function

Use the **capturegroups** inline function for more precise validation of fields featuring multi-line strings.

Procedure

1. Create a **metadata.yaml** file to match your use case. Use the following structure as an example:

```

apiVersion: v2
parts:
- name: Part1
  components:
    - name: Example
      allOf:
        - path: example.yaml
          config:
            perField:
              - pathToKey: spec.bigTextBlock ①
                inlineDiffFunc: capturegroups ②
  
```

- ① Specifies the field for inline validation.
- ② Enables inline validation using capture groups.

2. Use a regular expression to validate the field in the associated template:

```

apiVersion: v1
kind: ConfigMap
metadata:
  namespace: dashboard
  
```

```

data:
bigTextBlock: |-
  This static content outside of a capture group should match exactly.
  Here is a username capture group: (?<username>[a-z0-9]+).
  It should match this capture group: (?<username>[a-z0-9]+).

```

6.4.6. Configuring descriptions for the output

Each part, component, or template can include descriptions to provide additional context, instructions, or documentation links. These descriptions are helpful to convey why a specific template or structure is required.

Procedure

- Create a **metadata.yaml** file to match your use case. Use the following structure as an example:

```

apiVersion: v2
parts:
- name: Part1
  description: |-
    General text for every template under this part, unless overridden.
  components:
    - name: Component1
      # With no description set, this inherits the description from the part above.
      OneOf:
        - path: Template1.yaml
          # This inherits the component description, if set.
        - path: Template2.yaml
        - path: Template3.yaml
      description: |-
        This template has special instructions that don't apply to the others.
    - name: Component2
      description: |-
        This overrides the part text with something more specific.
        Multi-line text is supported, at all levels.
      allOf:
        - path: RequiredTemplate1.yaml
        - path: RequiredTemplate2.yaml
      description: |-
        Required for important reasons.
    - path: RequiredTemplate3.yaml

```

6.5. PERFORMING ADVANCED REFERENCE CONFIGURATION CUSTOMIZATION

For scenarios where you want to allow temporary deviations from the reference design, you can apply more advanced customizations.



WARNING

These customizations override the default matching process that the **cluster-compare** plugin uses during a comparison. Use caution when applying these advanced customizations as it can lead to unintended consequences, such as excluding consequential information from a cluster comparison.

Some advanced tasks to dynamically customize your reference configuration include the following:

- **Manual matching:** Configure a user configuration file to manually match a custom resource from the cluster to a template in the reference configuration.
- **Patching the reference:** Patch a reference to configure a reference configuration by using a patch option with the **cluster-compare** command.

6.5.1. Configuring manual matching between CRs and templates

In some cases, the **cluster-compare** plugin's default matching might not work as expected. You can manually define how a custom resource (CR) maps to a template by using a user configuration file.

By default, the plugin maps a CR to a template based on the **apiVersion**, **kind**, **name**, and **namespace** fields. However, multiple templates might match a single CR. For example, this can occur in the following scenarios:

- Multiple templates exist with the same **apiVersion**, **kind**, **name**, and **namespace** fields.
- Templates match any CR with a specific **apiVersion** and **kind**, regardless of its **namespace** or **name**.

When a CR matches multiple templates, the plugin uses a tie-breaking mechanism that selects the template with the fewest differences. To explicitly control which template the plugin chooses, you can create a user configuration YAML file that defines manual matching rules. You can pass this configuration file to the **cluster-compare** command to enforce the required template selection.

Procedure

- 1 Create a user configuration file to define the manual matching criteria:

Example `user-config.yaml` file

```
correlationSettings: ①
  manualCorrelation: ②
    correlationPairs: ③
      ptpt.openshift.io/v1_PtpConfig_openshift-ptp_grandmaster: optional/ptp-
      config/PtpOperatorConfig.yaml ④
      ptpt.openshift.io/v1_PtpOperatorConfig_openshift-ptp_default: optional/ptp-
      config/PtpOperatorConfig.yaml
```

- 1 The **correlationSettings** section contains the manual correlation settings.

- 2 The **manualCorrelation** section specifies that manual correlation is enabled.
 - 3 The **correlationPairs** section lists the CR and template pairs to manually match.
 - 4 Specifies the CR and template pair to match. The CR specification uses the following format: **<apiversion>_<kind>_<namespace>_<name>**. For cluster-scoped CRs that do not have a namespace, use the following format: **<apiversion>_<kind>_<name>**. The path to the template must be relative to the **metadata.yaml** file.
2. Reference the user configuration file in a **cluster-compare** command by running the following command:
- ```
$ oc cluster-compare -r <path_to_reference_config>/metadata.yaml -c <path_to_user_config>/user-config.yaml ①
```
- 1 Specify the **user-config.yaml** file by using the **-c** option.

## 6.5.2. Patching a reference configuration

In certain scenarios, you might need to patch the reference configuration to handle expected deviations in a cluster configuration. The plugin applies the patch during the comparison process, modifying the specified resource fields as defined in the patch file.

For example, you might need to temporarily patch a template because a cluster uses a deprecated field that is out-of-date with the latest reference configuration. Patched files are reported in the comparison output summary.

You can create a patch file in two ways:

- Use the **cluster-compare** plugin to generate a patch YAML file.
- Create your own patch file.

### 6.5.2.1. Using the cluster-compare plugin to generate a patch

You can use the **cluster-compare** plugin to generate a patch for specific template files. The plugin adjusts the template to ensure it matches with the cluster custom resource (CR). Any previously valid differences in the patched template are not reported. The plugin highlights the patched files in the output.

#### Procedure

- 1 Generate patches for templates by running the following command:

```
$ oc cluster-compare -r <path_to_reference_config>/metadata.yaml -o 'generate-patches' --override-reason "A valid reason for the override" --generate-override-for "<template1_path>" --generate-override-for "<template2_path>" > <path_to_patches_file>
```

- **-r** specifies the path to the **metadata.yaml** file of the reference configuration.
- **-o** specifies the output format. To generate a patch output, you must use the **generate-patches** value.

- **--override-reason** describes the reason for the patch.
- **--generate-override-for** specifies a path to the template that requires a patch.



#### NOTE

You must use a file path for the target template that is relative to the **metadata.yaml** file. For example, if the file path for the **metadata.yaml** file is **./compare/metadata.yaml**, a relative file path for the template might be **optional/my-template.yaml**.

- **<path\_to\_patches\_file>** specifies the filename and path for your patch.
2. Optional: Review the patch file before applying to the reference configuration:

#### Example patch-config file

```
- apiVersion: storage.k8s.io/v1
kind: StorageClass
name: crc-csi-hostpath-provisioner
patch: '{"provisioner":"kubebvirt.io.hostpath-provisioner"}' 1
reason: A valid reason for the override
templatePath: optional/local-storage-operator/StorageClass.yaml 2
type: mergepatch 3
```

- 1** The plugin patches the fields in the template to match the CR.
- 2** The path to the template.
- 3** The **mergepath** option merges the JSON into the target template. Unspecified fields remain unchanged.

3. Apply the patch to the reference configuration by running the following command:

```
$ oc cluster-compare -r <referenceConfigurationDirectory> -p <path_to_patches_file>
```

- **-r** specifies the path to the metadata.yaml file of the reference configuration.
- **-p** specifies the path to the patch file.

#### Example output

```
...
Cluster CR: storage.k8s.io/v1_StorageClass_crc-csi-hostpath-provisioner
Reference File: optional/local-storage-operator/StorageClass.yaml
Description: Component description
Diff Output: None
Patched with patch
Patch Reasons:
- A valid reason for the override
...
```

No CRs are unmatched to reference CRs  
 Metadata Hash:  
 bb2165004c496b32e0c8509428fb99c653c3cf4fba41196ea6821bd05c3083ab  
 Cluster CRs with patches applied: 1

### 6.5.2.2. Creating a patch file manually

You can write a patch file to handle expected deviations in a cluster configuration.



#### NOTE

Patches have three possible values for the **type** field:

- **mergepatch** - Merges the JSON into the target template. Unspecified fields remain unchanged.
- **rfc6902** - Merges the JSON in the target template using **add**, **remove**, **replace**, **move**, and **copy** operations. Each operation targets a specific path.
- **go-template** - Defines a Golang template. The plugin renders the template using the cluster custom resource (CR) as input and generates either a **mergepatch** or **rfc6902** patch for the target template.

The following example shows the same patch using all three different formats.

#### Procedure

1. Create a patch file to match your use case. Use the following structure as an example:

#### Example patch-config

```
- apiVersion: v1 ①
kind: Namespace
name: openshift-storage
reason: known deviation
templatePath: namespace.yaml
type: mergepatch
patch: '{"metadata":{"annotations":'
{"openshift.io/sa.scc.mcs":"s0:c29,c14","openshift.io/sa.scc.supplemental-
groups":"1000840000/10000","openshift.io/sa.scc.uid-
range":"1000840000/10000","reclaimspace.csiaddons.openshift.io/schedule":"@weekly","worklc-
ad.openshift.io/allowed":null},"labels":{"kubernetes.io/metadata.name":"openshift-
storage","olm.operatorgroup.uid/ffcf3f2d-3e37-4772-97bc-
983cdfce128b":""}, "openshift.io/cluster-monitoring":"false", "pod-
security.kubernetes.io/audit":"privileged", "pod-security.kubernetes.io/audit-
version":"v1.24", "pod-security.kubernetes.io/warn":"privileged", "pod-
security.kubernetes.io/warn-
version":"v1.24", "security.openshift.io/scc.podSecurityLabelSync":"true"}}, "spec":{"finalizers": [
["kubernetes"]]}
- name: openshift-storage
apiVersion: v1
kind: Namespace
templatePath: namespace.yaml
type: rfc6902
```

```

reason: known deviation
patch: [
 {"op": "add", "path": "/metadata/annotations/openshift.io~1sa.scc.mcs", "value": "s0:c29,c14"},
 {"op": "add", "path": "/metadata/annotations/openshift.io~1sa.scc.supplemental-groups", "value": "1000840000/10000"},
 {"op": "add", "path": "/metadata/annotations/openshift.io~1sa.scc.uid-range", "value": "1000840000/10000"},
 {"op": "add", "path": "/metadata/annotations/reclaimspace.csiaddons.openshift.io~1schedule", "value": "@weekly"},
 {"op": "remove", "path": "/metadata/annotations/workload.openshift.io~1allowed"},
 {"op": "add", "path": "/metadata/labels/kubernetes.io~1metadata.name", "value": "openshift-storage"},
 {"op": "add", "path": "/metadata/labels/olm.operatorgroup.uid~1ffcf3f2d-3e37-4772-97bc-983cdfce128b", "value": ""},
 {"op": "add", "path": "/metadata/labels/openshift.io~1cluster-monitoring", "value": "false"},
 {"op": "add", "path": "/metadata/labels/pod-security.kubernetes.io~1audit", "value": "privileged"},
 {"op": "add", "path": "/metadata/labels/pod-security.kubernetes.io~1audit-version", "value": "v1.24"},
 {"op": "add", "path": "/metadata/labels/pod-security.kubernetes.io~1warn", "value": "privileged"},
 {"op": "add", "path": "/metadata/labels/pod-security.kubernetes.io~1warn-version", "value": "v1.24"},
 {"op": "add", "path": "/metadata/labels/security.openshift.io~1scc.podSecurityLabelSync", "value": "true"},
 {"op": "add", "path": "/spec", "value": {"finalizers": ["kubernetes"]}}
]
-
apiVersion: v1
kind: Namespace
name: openshift-storage
reason: "known deviation"
templatePath: namespace.yaml
type: go-template
patch: |
{
 "type": "rfc6902",
 "patch": [
 {"op": "add", "path": "/metadata/annotations/openshift.io~1sa.scc.mcs", "value": "s0:c29,c14"},
 {"op": "add", "path": "/metadata/annotations/openshift.io~1sa.scc.supplemental-groups", "value": "1000840000/10000"},
 {"op": "add", "path": "/metadata/annotations/openshift.io~1sa.scc.uid-range", "value": "1000840000/10000"},
 {"op": "add", "path": "/metadata/annotations/reclaimspace.csiaddons.openshift.io~1schedule", "value": "@weekly"},
 {"op": "remove", "path": "/metadata/annotations/workload.openshift.io~1allowed"},
 {"op": "add", "path": "/metadata/labels/kubernetes.io~1metadata.name", "value": "openshift-storage"},
 {"op": "add", "path": "/metadata/labels/olm.operatorgroup.uid~1ffcf3f2d-3e37-4772-97bc-983cdfce128b", "value": ""},
 {"op": "add", "path": "/metadata/labels/openshift.io~1cluster-monitoring", "value": "false"},
 {"op": "add", "path": "/metadata/labels/pod-security.kubernetes.io~1audit", "value": "privileged"},
]
}

```

```

 "privileged"},

 {"op": "add", "path": "/metadata/labels/pod-security.kubernetes.io~1audit-version",

 "value": "v1.24"},

 {"op": "add", "path": "/metadata/labels/pod-security.kubernetes.io~1warn", "value":

 "privileged"},

 {"op": "add", "path": "/metadata/labels/pod-security.kubernetes.io~1warn-version",

 "value": "v1.24"},

 {"op": "add", "path":

 "/metadata/labels/security.openshift.io~1scc.podSecurityLabelSync", "value": "true"},

 {"op": "add", "path": "/spec", "value": {"finalizers": [{ .spec.finalizers | toJson }]} }
]

}

```

- 1 The patches uses the **kind**, **apiVersion**, **name**, and **namespace** fields to match the patch with the correct cluster CR.

2. Apply the patch to the reference configuration by running the following command:

```
$ oc cluster-compare -r <referenceConfigurationDirectory> -p <path_to_patches_file>
```

- **-r** specifies the path to the metadata.yaml file of the reference configuration.
- **p** specifies the path to the patch file.

#### Example output

```

...
Cluster CR: storage.k8s.io/v1_StorageClass_crc-csi-hostpath-provisioner
Reference File: namespace.yaml
Description: Component description
Diff Output: None
Patched with patch
Patch Reasons:
- known deviation
- known deviation
- known deviation
...
No CRs are unmatched to reference CRs
Metadata Hash:
bb2165004c496b32e0c8509428fb99c653c3cf4fba41196ea6821bd05c3083ab
Cluster CRs with patches applied: 1

```

## 6.6. TROUBLESHOOTING CLUSTER COMPARISONS

When using the **cluster-compare** plugin, you might see unexpected results, such as false positives or conflicts when multiple cluster custom resources (CRs) exist.

### 6.6.1. Troubleshooting false positives for missing resources

The plugin might report a missing resource even though the cluster custom resource (CR) is present in the cluster.

#### Procedure

1. Ensure you are using the latest version of the **cluster-compare** plugin. For more information, see "Installing the cluster-compare plugin".
2. Ensure you are using the most up-to-date version of the reference configuration.
3. Ensure that template has the same **apiVersion**, **kind**, **name**, and **namespace** fields as the cluster CR.

#### 6.6.2. Troubleshooting multiple template matches for the same CR

In some cases, more than one cluster CR can match a template because they feature the same **apiVersion**, **namespace**, and **kind**. The plugin's default matching compares the CR that features the least differences.

You can optionally configure your reference configuration to avoid this situation.

#### Procedure

1. Ensure the templates feature distinct **apiVersion**, **namespace**, and **kind** values to ensure no duplicate template matching.
2. Use a user configuration file to manually match a template to a CR. For more information, see "Configuring manual matching between CRs and templates".

#### 6.6.3. Additional resources

- [Installing the cluster-compare plugin](#)
- [Configuring manual matching between CRs and templates](#)

# CHAPTER 7. PLANNING YOUR ENVIRONMENT ACCORDING TO OBJECT MAXIMUMS

Consider the following tested object maximums when you plan your OpenShift Container Platform cluster.

These guidelines are based on the largest possible cluster. For smaller clusters, the maximums are lower. There are many factors that influence the stated thresholds, including the etcd version or storage data format.

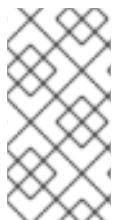
In most cases, exceeding these numbers results in lower overall performance. It does not necessarily mean that the cluster will fail.



## WARNING

Clusters that experience rapid change, such as those with many starting and stopping pods, can have a lower practical maximum size than documented.

## 7.1. OPENSIFT CONTAINER PLATFORM TESTED CLUSTER MAXIMUMS FOR MAJOR RELEASES



## NOTE

Red Hat does not provide direct guidance on sizing your OpenShift Container Platform cluster. This is because determining whether your cluster is within the supported bounds of OpenShift Container Platform requires careful consideration of all the multidimensional factors that limit the cluster scale.

OpenShift Container Platform supports tested cluster maximums rather than absolute cluster maximums. Not every combination of OpenShift Container Platform version, control plane workload, and network plugin are tested, so the following table does not represent an absolute expectation of scale for all deployments. It might not be possible to scale to a maximum on all dimensions simultaneously. The table contains tested maximums for specific workload and deployment configurations, and serves as a scale guide as to what can be expected with similar deployments.

| Maximum type            | 4.x tested maximum         |
|-------------------------|----------------------------|
| Number of nodes         | 2,000 [1]                  |
| Number of pods [2]      | 150,000                    |
| Number of pods per node | 2,500 [3]                  |
| Number of pods per core | There is no default value. |

| Maximum type                                          | 4.x tested maximum                                                     |
|-------------------------------------------------------|------------------------------------------------------------------------|
| Number of namespaces [4]                              | 10,000                                                                 |
| Number of builds                                      | 10,000 (Default pod RAM 512 Mi) – Source-to-Image (S2I) build strategy |
| Number of pods per namespace [5]                      | 25,000                                                                 |
| Number of routes and back ends per Ingress Controller | 2,000 per router                                                       |
| Number of secrets                                     | 80,000                                                                 |
| Number of config maps                                 | 90,000                                                                 |
| Number of services [6]                                | 10,000                                                                 |
| Number of services per namespace                      | 5,000                                                                  |
| Number of back-ends per service                       | 5,000                                                                  |
| Number of deployments per namespace [5]               | 2,000                                                                  |
| Number of build configs                               | 12,000                                                                 |
| Number of custom resource definitions (CRD)           | 1,024 [7]                                                              |

1. Pause pods were deployed to stress the control plane components of OpenShift Container Platform at 2000 node scale. The ability to scale to similar numbers will vary depending upon specific deployment and workload parameters.
2. The pod count displayed here is the number of test pods. The actual number of pods depends on the application's memory, CPU, and storage requirements.
3. This was tested on a cluster with 31 servers: 3 control planes, 2 infrastructure nodes, and 26 worker nodes. If you need 2,500 user pods, you need both a **hostPrefix** of **20**, which allocates a network large enough for each node to contain more than 2000 pods, and a custom kubelet config with **maxPods** set to **2500**. For more information, see [Running 2500 pods per node on OCP 4.13](#).
4. When there are a large number of active projects, etcd might suffer from poor performance if the keyspace grows excessively large and exceeds the space quota. Periodic maintenance of etcd, including defragmentation, is highly recommended to free etcd storage.
5. There are several control loops in the system that must iterate over all objects in a given namespace as a reaction to some changes in state. Having a large number of objects of a given type in a single namespace can make those loops expensive and slow down processing given

state changes. The limit assumes that the system has enough CPU, memory, and disk to satisfy the application requirements.

6. Each service port and each service back-end has a corresponding entry in **iptables**. The number of back-ends of a given service impact the size of the **Endpoints** objects, which impacts the size of data that is being sent all over the system.
7. Tested on a cluster with 29 servers: 3 control planes, 2 infrastructure nodes, and 24 worker nodes. The cluster had 500 namespaces. OpenShift Container Platform has a limit of 1,024 total custom resource definitions (CRD), including those installed by OpenShift Container Platform, products integrating with OpenShift Container Platform and user-created CRDs. If there are more than 1,024 CRDs created, then there is a possibility that **oc** command requests might be throttled.

### 7.1.1. Example scenario

As an example, 500 worker nodes (m5.2xl) were tested, and are supported, using OpenShift Container Platform 4.18, the OVN-Kubernetes network plugin, and the following workload objects:

- 200 namespaces, in addition to the defaults
- 60 pods per node; 30 server and 30 client pods (30k total)
- 57 image streams/ns (11.4k total)
- 15 services/ns backed by the server pods (3k total)
- 15 routes/ns backed by the previous services (3k total)
- 20 secrets/ns (4k total)
- 10 config maps/ns (2k total)
- 6 network policies/ns, including deny-all, allow-from ingress and intra-namespace rules
- 57 builds/ns

The following factors are known to affect cluster workload scaling, positively or negatively, and should be factored into the scale numbers when planning a deployment. For additional information and guidance, contact your sales representative or [Red Hat support](#).

- Number of pods per node
- Number of containers per pod
- Type of probes used (for example, liveness/readiness, exec/http)
- Number of network policies
- Number of projects, or namespaces
- Number of image streams per project
- Number of builds per project
- Number of services/endpoints and type

- Number of routes
- Number of shards
- Number of secrets
- Number of config maps
- Rate of API calls, or the cluster “churn”, which is an estimation of how quickly things change in the cluster configuration.
  - Prometheus query for pod creation requests per second over 5 minute windows:  
`sum(irate(apiserver_request_count{resource="pods",verb="POST"}[5m]))`
  - Prometheus query for all API requests per second over 5 minute windows:  
`sum(irate(apiserver_request_count{}[5m]))`
- Cluster node resource consumption of CPU
- Cluster node resource consumption of memory

## 7.2. OPENSIFT CONTAINER PLATFORM ENVIRONMENT AND CONFIGURATION ON WHICH THE CLUSTER MAXIMUMS ARE TESTED

### 7.2.1. AWS cloud platform

| Node                       | Flavor      | vCPU | RAM(GiB) | Disk type | Disk size(GiB) /IOPS        | Count | Region    |
|----------------------------|-------------|------|----------|-----------|-----------------------------|-------|-----------|
| Control plane/etc<br>d [1] | r5.4xlarge  | 16   | 128      | gp3       | 220                         | 3     | us-west-2 |
| Infra [2]                  | m5.12xlarge | 48   | 192      | gp3       | 100                         | 3     | us-west-2 |
| Workload [3]               | m5.4xlarge  | 16   | 64       | gp3       | 500 [4]                     | 1     | us-west-2 |
| Compute                    | m5.2xlarge  | 8    | 32       | gp3       | 100<br>3/25/250<br>/500 [5] | 250   | us-west-2 |

1. gp3 disks with a baseline performance of 3000 IOPS and 125 MiB per second are used for control plane/etc<sub>d</sub> nodes because etcd is latency sensitive. gp3 volumes do not use burst performance.
2. Infra nodes are used to host Monitoring, Ingress, and Registry components to ensure they have enough resources to run at large scale.
3. Workload node is dedicated to run performance and scalability workload generators.

4. Larger disk size is used so that there is enough space to store the large amounts of data that is collected during the performance and scalability test run.
5. Cluster is scaled in iterations and performance and scalability tests are executed at the specified node counts.

### 7.2.2. IBM Power platform

| Node                    | vCPU | RAM(GiB) | Disk type | Disk size(GiB)/IOS    | Count        |
|-------------------------|------|----------|-----------|-----------------------|--------------|
| Control plane/etcdb [1] | 16   | 32       | io1       | 120 / 10 IOPS per GiB | 3            |
| Infra [2]               | 16   | 64       | gp2       | 120                   | 2            |
| Workload [3]            | 16   | 256      | gp2       | 120 [4]               | 1            |
| Compute                 | 16   | 64       | gp2       | 120                   | 2 to 100 [5] |

1. io1 disks with 120 / 10 IOPS per GiB are used for control plane/etcdb nodes as etcd is I/O intensive and latency sensitive.
2. Infra nodes are used to host Monitoring, Ingress, and Registry components to ensure they have enough resources to run at large scale.
3. Workload node is dedicated to run performance and scalability workload generators.
4. Larger disk size is used so that there is enough space to store the large amounts of data that is collected during the performance and scalability test run.
5. Cluster is scaled in iterations.

### 7.2.3. IBM Z platform

| Node                      | vCPU [4] | RAM(GiB)[5] | Disk type | Disk size(GiB)/IOS | Count                                         |
|---------------------------|----------|-------------|-----------|--------------------|-----------------------------------------------|
| Control plane/etcdb [1,2] | 8        | 32          | ds8k      | 300 / LCU 1        | 3                                             |
| Compute [1,3]             | 8        | 32          | ds8k      | 150 / LCU 2        | 4 nodes (scaled to 100/250/500 pods per node) |

1. Nodes are distributed between two logical control units (LCUs) to optimize disk I/O load of the control plane/etc<sub>d</sub> nodes as etc<sub>d</sub> is I/O intensive and latency sensitive. Etcd I/O demand should not interfere with other workloads.
2. Four compute nodes are used for the tests running several iterations with 100/250/500 pods at the same time. First, idling pods were used to evaluate if pods can be instanced. Next, a network and CPU demanding client/server workload were used to evaluate the stability of the system under stress. Client and server pods were pairwise deployed and each pair was spread over two compute nodes.
3. No separate workload node was used. The workload simulates a microservice workload between two compute nodes.
4. Physical number of processors used is six Integrated Facilities for Linux (IFLs).
5. Total physical memory used is 512 GiB.

## 7.3. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO TESTED CLUSTER MAXIMUMS



### IMPORTANT

Oversubscribing the physical resources on a node affects resource guarantees the Kubernetes scheduler makes during pod placement. Learn what measures you can take to avoid memory swapping.

Some of the tested maximums are stretched only in a single dimension. They will vary when many objects are running on the cluster.

The numbers noted in this documentation are based on Red Hat's test methodology, setup, configuration, and tunings. These numbers can vary based on your own individual setup and environments.

While planning your environment, determine how many pods are expected to fit per node:

$$\text{required pods per cluster / pods per node} = \text{total number of nodes needed}$$

The default maximum number of pods per node is 250. However, the number of pods that fit on a node is dependent on the application itself. Consider the application's memory, CPU, and storage requirements, as described in "How to plan your environment according to application requirements".

### Example scenario

If you want to scope your cluster for 2200 pods per cluster, you would need at least five nodes, assuming that there are 500 maximum pods per node:

$$2200 / 500 = 4.4$$

If you increase the number of nodes to 20, then the pod distribution changes to 110 pods per node:

$$2200 / 20 = 110$$

Where:

required pods per cluster / total number of nodes = expected pods per node

OpenShift Container Platform comes with several system pods, such as OVN-Kubernetes, DNS, Operators, and others, which run across every worker node by default. Therefore, the result of the above formula can vary.

## 7.4. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO APPLICATION REQUIREMENTS

Consider an example application environment:

| Pod type   | Pod quantity | Max memory | CPU cores | Persistent storage |
|------------|--------------|------------|-----------|--------------------|
| apache     | 100          | 500 MB     | 0.5       | 1 GB               |
| node.js    | 200          | 1 GB       | 1         | 1 GB               |
| postgresql | 100          | 1 GB       | 2         | 10 GB              |
| JBoss EAP  | 100          | 1 GB       | 1         | 1 GB               |

Extrapolated requirements: 550 CPU cores, 450GB RAM, and 1.4TB storage.

Instance size for nodes can be modulated up or down, depending on your preference. Nodes are often resource overcommitted. In this deployment scenario, you can choose to run additional smaller nodes or fewer larger nodes to provide the same amount of resources. Factors such as operational agility and cost-per-instance should be considered.

| Node type        | Quantity | CPUs | RAM (GB) |
|------------------|----------|------|----------|
| Nodes (option 1) | 100      | 4    | 16       |
| Nodes (option 2) | 50       | 8    | 32       |
| Nodes (option 3) | 25       | 16   | 64       |

Some applications lend themselves well to overcommitted environments, and some do not. Most Java applications and applications that use huge pages are examples of applications that would not allow for overcommitment. That memory can not be used for other applications. In the example above, the environment would be roughly 30 percent overcommitted, a common ratio.

The application pods can access a service either by using environment variables or DNS. If using environment variables, for each active service the variables are injected by the kubelet when a pod is run on a node. A cluster-aware DNS server watches the Kubernetes API for new services and creates a set of DNS records for each one. If DNS is enabled throughout your cluster, then all pods should automatically be able to resolve services by their DNS name. Service discovery using DNS can be used in case you must go beyond 5000 services. When using environment variables for service discovery, the

argument list exceeds the allowed length after 5000 services in a namespace, then the pods and deployments will start failing. Disable the service links in the deployment's service specification file to overcome this:

```

apiVersion: template.openshift.io/v1
kind: Template
metadata:
 name: deployment-config-template
 creationTimestamp:
 annotations:
 description: This template will create a deploymentConfig with 1 replica, 4 env vars and a service.
 tags: ""
objects:
- apiVersion: apps.openshift.io/v1
 kind: DeploymentConfig
 metadata:
 name: deploymentconfig${IDENTIFIER}
 spec:
 template:
 metadata:
 labels:
 name: replicationcontroller${IDENTIFIER}
 spec:
 enableServiceLinks: false
 containers:
- name: pause${IDENTIFIER}
 image: "${IMAGE}"
 ports:
- containerPort: 8080
 protocol: TCP
 env:
- name: ENVVAR1_${IDENTIFIER}
 value: "${ENV_VALUE}"
- name: ENVVAR2_${IDENTIFIER}
 value: "${ENV_VALUE}"
- name: ENVVAR3_${IDENTIFIER}
 value: "${ENV_VALUE}"
- name: ENVVAR4_${IDENTIFIER}
 value: "${ENV_VALUE}"
 resources: {}
 imagePullPolicy: IfNotPresent
 capabilities: {}
 securityContext:
 capabilities: {}
 privileged: false
 restartPolicy: Always
 serviceAccount: ""
 replicas: 1
 selector:
 name: replicationcontroller${IDENTIFIER}
 triggers:
- type: ConfigChange
 strategy:
 type: Rolling
- apiVersion: v1
```

```

Kind: Service
metadata:
 name: service${IDENTIFIER}
spec:
 selector:
 name: replicationcontroller${IDENTIFIER}
 ports:
 - name: serviceport${IDENTIFIER}
 protocol: TCP
 port: 80
 targetPort: 8080
 clusterIP: ""
 type: ClusterIP
 sessionAffinity: None
status:
 loadBalancer: {}
parameters:
- name: IDENTIFIER
 description: Number to append to the name of resources
 value: '1'
 required: true
- name: IMAGE
 description: Image to use for deploymentConfig
 value: gcr.io/google-containers/pause-amd64:3.0
 required: false
- name: ENV_VALUE
 description: Value to use for environment variables
 generate: expression
 from: "[A-Za-z0-9]{255}"
 required: false
labels:
 template: deployment-config-template

```

The number of application pods that can run in a namespace is dependent on the number of services and the length of the service name when the environment variables are used for service discovery.

**ARG\_MAX** on the system defines the maximum argument length for a new process and it is set to 2097152 bytes (2 MiB) by default. The Kubelet injects environment variables in to each pod scheduled to run in the namespace including:

- <SERVICE\_NAME>\_SERVICE\_HOST=<IP>
- <SERVICE\_NAME>\_SERVICE\_PORT=<PORT>
- <SERVICE\_NAME>\_PORT=tcp://<IP>:<PORT>
- <SERVICE\_NAME>\_PORT\_<PORT>\_TCP=tcp://<IP>:<PORT>
- <SERVICE\_NAME>\_PORT\_<PORT>\_TCP\_PROTO=tcp
- <SERVICE\_NAME>\_PORT\_<PORT>\_TCP\_PORT=<PORT>
- <SERVICE\_NAME>\_PORT\_<PORT>\_TCP\_ADDR=<ADDR>

The pods in the namespace will start to fail if the argument length exceeds the allowed value and the number of characters in a service name impacts it. For example, in a namespace with 5000 services, the limit on the service name is 33 characters, which enables you to run 5000 pods in the namespace.

# CHAPTER 8. USING QUOTAS AND LIMIT RANGES

A resource quota, defined by a **ResourceQuota** object, provides constraints that limit aggregate resource consumption per project. It can limit the quantity of objects that can be created in a project by type, as well as the total amount of compute resources and storage that may be consumed by resources in that project.

Using quotas and limit ranges, cluster administrators can set constraints to limit the number of objects or amount of compute resources that are used in your project. This helps cluster administrators better manage and allocate resources across all projects, and ensure that no projects are using more than is appropriate for the cluster size.



## IMPORTANT

Quotas are set by cluster administrators and are scoped to a given project. OpenShift Container Platform project owners can change quotas for their project, but not limit ranges. OpenShift Container Platform users cannot modify quotas or limit ranges.

The following sections help you understand how to check on your quota and limit range settings, what sorts of things they can constrain, and how you can request or limit compute resources in your own pods and containers.

## 8.1. RESOURCES MANAGED BY QUOTA

A resource quota, defined by a **ResourceQuota** object, provides constraints that limit aggregate resource consumption per project. It can limit the quantity of objects that can be created in a project by type, as well as the total amount of compute resources and storage that may be consumed by resources in that project.

The following describes the set of compute resources and object types that may be managed by a quota.



## NOTE

A pod is in a terminal state if **status.phase** is **Failed** or **Succeeded**.

**Table 8.1. Compute resources managed by quota**

| Resource Name | Description                                                                                                                                                                               |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>cpu</b>    | The sum of CPU requests across all pods in a non-terminal state cannot exceed this value. <b>cpu</b> and <b>requests.cpu</b> are the same value and can be used interchangeably.          |
| <b>memory</b> | The sum of memory requests across all pods in a non-terminal state cannot exceed this value. <b>memory</b> and <b>requests.memory</b> are the same value and can be used interchangeably. |

| Resource Name                     | Description                                                                                                                                                                                                                                                                                                                                                    |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ephemeral-storage</b>          | The sum of local ephemeral storage requests across all pods in a non-terminal state cannot exceed this value. <b>ephemeral-storage</b> and <b>requests.ephemeral-storage</b> are the same value and can be used interchangeably. This resource is available only if you enabled the ephemeral storage technology preview. This feature is disabled by default. |
| <b>requests.cpu</b>               | The sum of CPU requests across all pods in a non-terminal state cannot exceed this value. <b>cpu</b> and <b>requests.cpu</b> are the same value and can be used interchangeably.                                                                                                                                                                               |
| <b>requests.memory</b>            | The sum of memory requests across all pods in a non-terminal state cannot exceed this value. <b>memory</b> and <b>requests.memory</b> are the same value and can be used interchangeably.                                                                                                                                                                      |
| <b>requests.ephemeral-storage</b> | The sum of ephemeral storage requests across all pods in a non-terminal state cannot exceed this value. <b>ephemeral-storage</b> and <b>requests.ephemeral-storage</b> are the same value and can be used interchangeably. This resource is available only if you enabled the ephemeral storage technology preview. This feature is disabled by default.       |
| <b>limits.cpu</b>                 | The sum of CPU limits across all pods in a non-terminal state cannot exceed this value.                                                                                                                                                                                                                                                                        |
| <b>limits.memory</b>              | The sum of memory limits across all pods in a non-terminal state cannot exceed this value.                                                                                                                                                                                                                                                                     |
| <b>limits.ephemeral-storage</b>   | The sum of ephemeral storage limits across all pods in a non-terminal state cannot exceed this value. This resource is available only if you enabled the ephemeral storage technology preview. This feature is disabled by default.                                                                                                                            |

Table 8.2. Storage resources managed by quota

| Resource Name                                                                  | Description                                                                                                                                |
|--------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <b>requests.storage</b>                                                        | The sum of storage requests across all persistent volume claims in any state cannot exceed this value.                                     |
| <b>persistentvolumeclaims</b>                                                  | The total number of persistent volume claims that can exist in the project.                                                                |
| <b>&lt;storage-class-name&gt;.storageclass.storage.k8s.io/requests.storage</b> | The sum of storage requests across all persistent volume claims in any state that have a matching storage class, cannot exceed this value. |

| Resource Name                                                                        | Description                                                                                               |
|--------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------|
| <b>&lt;storage-class-name&gt;.storageclass.storage.k8s.io/persistentvolumeclaims</b> | The total number of persistent volume claims with a matching storage class that can exist in the project. |

Table 8.3. Object counts managed by quota

| Resource Name                    | Description                                                                     |
|----------------------------------|---------------------------------------------------------------------------------|
| <b>pods</b>                      | The total number of pods in a non-terminal state that can exist in the project. |
| <b>replicationcontrollers</b>    | The total number of replication controllers that can exist in the project.      |
| <b>resourcequotas</b>            | The total number of resource quotas that can exist in the project.              |
| <b>services</b>                  | The total number of services that can exist in the project.                     |
| <b>secrets</b>                   | The total number of secrets that can exist in the project.                      |
| <b>configmaps</b>                | The total number of <b>ConfigMap</b> objects that can exist in the project.     |
| <b>persistentvolumeclaims</b>    | The total number of persistent volume claims that can exist in the project.     |
| <b>openshift.io/imagestreams</b> | The total number of image streams that can exist in the project.                |

You can configure an object count quota for these standard namespaced resource types using the `count/<resource>.<group>` syntax.

\$ oc create quota <name> --hard=count/<resource>.<group>=<quota> ①

- ① `<resource>` is the name of the resource, and `<group>` is the API group, if applicable. Use the **kubectl api-resources** command for a list of resources and their associated API groups.

### 8.1.1. Setting resource quota for extended resources

Overcommitment of resources is not allowed for extended resources, so you must specify **requests** and **limits** for the same extended resource in a quota. Currently, only quota items with the prefix **requests**. are allowed for extended resources. The following is an example scenario of how to set resource quota for the GPU resource **nvidia.com/gpu**.

#### Procedure

- To determine how many GPUs are available on a node in your cluster, use the following command:

```
$ oc describe node ip-172-31-27-209.us-west-2.compute.internal | egrep 'Capacity|Allocatable|gpu'
```

#### Example output

```
openshift.com/gpu-accelerator=true
Capacity:
nvidia.com/gpu: 2
Allocatable:
nvidia.com/gpu: 2
nvidia.com/gpu: 0 0
```

In this example, 2 GPUs are available.

- Use this command to set a quota in the namespace **nvidia**. In this example, the quota is **1**:

```
$ cat gpu-quota.yaml
```

#### Example output

```
apiVersion: v1
kind: ResourceQuota
metadata:
 name: gpu-quota
 namespace: nvidia
spec:
 hard:
 requests.nvidia.com/gpu: 1
```

- Create the quota with the following command:

```
$ oc create -f gpu-quota.yaml
```

#### Example output

```
resourcequota/gpu-quota created
```

- Verify that the namespace has the correct quota set using the following command:

```
$ oc describe quota gpu-quota -n nvidia
```

#### Example output

| Name:                   | gpu-quota |
|-------------------------|-----------|
| Namespace:              | nvidia    |
| Resource                | Used Hard |
| -----                   | -----     |
| requests.nvidia.com/gpu | 0 1       |

5. Run a pod that asks for a single GPU with the following command:

```
$ oc create pod gpu-pod.yaml
```

#### Example output

```
apiVersion: v1
kind: Pod
metadata:
 generateName: gpu-pod-s46h7
 namespace: nvidia
spec:
 restartPolicy: OnFailure
 containers:
 - name: rhel7-gpu-pod
 image: rhel7
 env:
 - name: NVIDIA_VISIBLE_DEVICES
 value: all
 - name: NVIDIA_DRIVER_CAPABILITIES
 value: "compute,utility"
 - name: NVIDIA_REQUIRE_CUDA
 value: "cuda>=5.0"
 command: ["sleep"]
 args: ["infinity"]
 resources:
 limits:
 nvidia.com/gpu: 1
```

6. Verify that the pod is running bwith the following command:

```
$ oc get pods
```

#### Example output

| NAME          | READY | STATUS  | RESTARTS | AGE |
|---------------|-------|---------|----------|-----|
| gpu-pod-s46h7 | 1/1   | Running | 0        | 1m  |

7. Verify that the quota **Used** counter is correct by running the following command:

```
$ oc describe quota gpu-quota -n nvidia
```

#### Example output

| Name:                   | gpu-quota |
|-------------------------|-----------|
| Namespace:              | nvidia    |
| Resource                | Used Hard |
| -----                   | ---- ---- |
| requests.nvidia.com/gpu | 1 1       |

- Using the following command, attempt to create a second GPU pod in the **nvidia** namespace. This is technically available on the node because it has 2 GPUs:

```
$ oc create -f gpu-pod.yaml
```

### Example output

```
Error from server (Forbidden): error when creating "gpu-pod.yaml": pods "gpu-pod-f7z2w" is forbidden: exceeded quota: gpu-quota, requested: requests.nvidia.com/gpu=1, used: requests.nvidia.com/gpu=1, limited: requests.nvidia.com/gpu=1
```

This **Forbidden** error message occurs because you have a quota of 1 GPU and this pod tried to allocate a second GPU, which exceeds its quota.

## 8.1.2. Quota scopes

Each quota can have an associated set of scopes. A quota only measures usage for a resource if it matches the intersection of enumerated scopes.

Adding a scope to a quota restricts the set of resources to which that quota can apply. Specifying a resource outside of the allowed set results in a validation error.

| Scope                 | Description                                                                                   |
|-----------------------|-----------------------------------------------------------------------------------------------|
| <b>Terminating</b>    | Match pods where <b>spec.activeDeadlineSeconds &gt;= 0</b> .                                  |
| <b>NotTerminating</b> | Match pods where <b>spec.activeDeadlineSeconds</b> is <b>nil</b> .                            |
| <b>BestEffort</b>     | Match pods that have best effort quality of service for either <b>cpu</b> or <b>memory</b> .  |
| <b>notBestEffort</b>  | Match pods that do not have best effort quality of service for <b>cpu</b> and <b>memory</b> . |

A **BestEffort** scope restricts a quota to limiting the following resources:

- **pods**

A **Terminating**, **NotTerminating**, and **NotBestEffort** scope restricts a quota to tracking the following resources:

- **pods**
- **memory**
- **requests.memory**
- **limits.memory**
- **cpu**
- **requests.cpu**

- **limits.cpu**
- **ephemeral-storage**
- **requests.ephemeral-storage**
- **limits.ephemeral-storage**



#### NOTE

Ephemeral storage requests and limits apply only if you enabled the ephemeral storage technology preview. This feature is disabled by default.

### Additional resources

See [Resources managed by quotas](#) for more on compute resources.

See [Quality of Service Classes](#) for more on committing compute resources.

## 8.2. ADMIN QUOTA USAGE

### 8.2.1. Quota enforcement

After a resource quota for a project is first created, the project restricts the ability to create any new resources that can violate a quota constraint until it has calculated updated usage statistics.

After a quota is created and usage statistics are updated, the project accepts the creation of new content. When you create or modify resources, your quota usage is incremented immediately upon the request to create or modify the resource.

When you delete a resource, your quota use is decremented during the next full recalculation of quota statistics for the project.

A configurable amount of time determines how long it takes to reduce quota usage statistics to their current observed system value.

If project modifications exceed a quota usage limit, the server denies the action, and an appropriate error message is returned to the user explaining the quota constraint violated, and what their currently observed usage stats are in the system.

### 8.2.2. Requests compared to limits

When allocating compute resources by quota, each container can specify a request and a limit value each for CPU, memory, and ephemeral storage. Quotas can restrict any of these values.

If the quota has a value specified for **requests.cpu** or **requests.memory**, then it requires that every incoming container make an explicit request for those resources. If the quota has a value specified for **limits.cpu** or **limits.memory**, then it requires that every incoming container specify an explicit limit for those resources.

### 8.2.3. Sample resource quota definitions

#### Example core-object-counts.yaml

```

apiVersion: v1
kind: ResourceQuota
metadata:
 name: core-object-counts
spec:
 hard:
 configmaps: "10" ①
 persistentvolumeclaims: "4" ②
 replicationcontrollers: "20" ③
 secrets: "10" ④
 services: "10" ⑤

```

- ① The total number of **ConfigMap** objects that can exist in the project.
- ② The total number of persistent volume claims (PVCs) that can exist in the project.
- ③ The total number of replication controllers that can exist in the project.
- ④ The total number of secrets that can exist in the project.
- ⑤ The total number of services that can exist in the project.

#### Example openshift-object-counts.yaml

```

apiVersion: v1
kind: ResourceQuota
metadata:
 name: openshift-object-counts
spec:
 hard:
 openshift.io/imagestreams: "10" ①

```

- ① The total number of image streams that can exist in the project.

#### Example compute-resources.yaml

```

apiVersion: v1
kind: ResourceQuota
metadata:
 name: compute-resources
spec:
 hard:
 pods: "4" ①
 requests.cpu: "1" ②
 requests.memory: 1Gi ③
 requests.ephemeral-storage: 2Gi ④
 limits.cpu: "2" ⑤
 limits.memory: 2Gi ⑥
 limits.ephemeral-storage: 4Gi ⑦

```

- ① The total number of pods in a non-terminal state that can exist in the project.

- 2 Across all pods in a non-terminal state, the sum of CPU requests cannot exceed 1 core.
- 3 Across all pods in a non-terminal state, the sum of memory requests cannot exceed 1Gi.
- 4 Across all pods in a non-terminal state, the sum of ephemeral storage requests cannot exceed 2Gi.
- 5 Across all pods in a non-terminal state, the sum of CPU limits cannot exceed 2 cores.
- 6 Across all pods in a non-terminal state, the sum of memory limits cannot exceed 2Gi.
- 7 Across all pods in a non-terminal state, the sum of ephemeral storage limits cannot exceed 4Gi.

### Example `besteffort.yaml`

```
apiVersion: v1
kind: ResourceQuota
metadata:
 name: besteffort
spec:
 hard:
 pods: "1" ①
 scopes:
 - BestEffort ②
```

- 1 The total number of pods in a non-terminal state with **BestEffort** quality of service that can exist in the project.
- 2 Restricts the quota to only matching pods that have **BestEffort** quality of service for either memory or CPU.

### Example `compute-resources-long-running.yaml`

```
apiVersion: v1
kind: ResourceQuota
metadata:
 name: compute-resources-long-running
spec:
 hard:
 pods: "4" ①
 limits.cpu: "4" ②
 limits.memory: "2Gi" ③
 limits.ephemeral-storage: "4Gi" ④
 scopes:
 - NotTerminating ⑤
```

- 1 The total number of pods in a non-terminal state.
- 2 Across all pods in a non-terminal state, the sum of CPU limits cannot exceed this value.
- 3 Across all pods in a non-terminal state, the sum of memory limits cannot exceed this value.
- 4 Across all pods in a non-terminal state, the sum of ephemeral storage limits cannot exceed this value.

- 5 Restricts the quota to only matching pods where **spec.activeDeadlineSeconds** is set to **nil**. Build pods will fall under **NotTerminating** unless the **RestartNever** policy is applied.

### Example compute-resources-time-bound.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
 name: compute-resources-time-bound
spec:
 hard:
 pods: "2" ①
 limits.cpu: "1" ②
 limits.memory: "1Gi" ③
 limits.ephemeral-storage: "1Gi" ④
 scopes:
 - Terminating ⑤
```

- ① The total number of pods in a non-terminal state.
- ② Across all pods in a non-terminal state, the sum of CPU limits cannot exceed this value.
- ③ Across all pods in a non-terminal state, the sum of memory limits cannot exceed this value.
- ④ Across all pods in a non-terminal state, the sum of ephemeral storage limits cannot exceed this value.
- ⑤ Restricts the quota to only matching pods where **spec.activeDeadlineSeconds >=0**. For example, this quota would charge for build pods, but not long running pods such as a web server or database.

### Example storage-consumption.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
 name: storage-consumption
spec:
 hard:
 persistentvolumeclaims: "10" ①
 requests.storage: "50Gi" ②
 gold.storageclass.storage.k8s.io/requests.storage: "10Gi" ③
 silver.storageclass.storage.k8s.io/requests.storage: "20Gi" ④
 silver.storageclass.storage.k8s.io/persistentvolumeclaims: "5" ⑤
 bronze.storageclass.storage.k8s.io/requests.storage: "0" ⑥
 bronze.storageclass.storage.k8s.io/persistentvolumeclaims: "0" ⑦
```

- ① The total number of persistent volume claims in a project
- ② Across all persistent volume claims in a project, the sum of storage requested cannot exceed this value.
- ③ Across all persistent volume claims in a project, the sum of storage requested in the gold storage

class cannot exceed this value.

- 4 Across all persistent volume claims in a project, the sum of storage requested in the silver storage class cannot exceed this value.
- 5 Across all persistent volume claims in a project, the total number of claims in the silver storage class cannot exceed this value.
- 6 Across all persistent volume claims in a project, the sum of storage requested in the bronze storage class cannot exceed this value. When this is set to **0**, it means bronze storage class cannot request storage.
- 7 Across all persistent volume claims in a project, the sum of storage requested in the bronze storage class cannot exceed this value. When this is set to **0**, it means bronze storage class cannot create claims.

#### 8.2.4. Creating a quota

To create a quota, first define the quota in a file. Then use that file to apply it to a project. See the Additional resources section for a link describing this.

```
$ oc create -f <resource_quota_definition> [-n <project_name>]
```

Here is an example using the **core-object-counts.yaml** resource quota definition and the **demoproject** project name:

```
$ oc create -f core-object-counts.yaml -n demoproject
```

#### 8.2.5. Creating object count quotas

You can create an object count quota for all OpenShift Container Platform standard namespaced resource types, such as **BuildConfig**, and **DeploymentConfig**. An object quota count places a defined quota on all standard namespaced resource types.

When using a resource quota, an object is charged against the quota if it exists in server storage. These types of quotas are useful to protect against exhaustion of storage resources.

To configure an object count quota for a resource, run the following command:

```
$ oc create quota <name> --hard=count/<resource>.<group>=<quota>,count/<resource>.<group>=<quota>
```

#### Example showing object count quota:

```
$ oc create quota test --
hard=count/deployments.extensions=2,count/replicasets.extensions=4,count/pods=3,count/secrets=4
resourcequota "test" created
```

```
$ oc describe quota test
Name: test
Namespace: quota
Resource Used Hard

```

```
count/deployments.extensions 0 2
count/pods 0 3
count/replicasets.extensions 0 4
count/secrets 0 4
```

This example limits the listed resources to the hard limit in each project in the cluster.

### 8.2.6. Viewing a quota

You can view usage statistics related to any hard limits defined in a project's quota by navigating in the web console to the project's **Quota** page.

You can also use the CLI to view quota details:

1. First, get the list of quotas defined in the project. For example, for a project called **demoproject**:

```
$ oc get quota -n demoproject
NAME AGE
besteffort 11m
compute-resources 2m
core-object-counts 29m
```

2. Describe the quota you are interested in, for example the **core-object-counts** quota:

```
$ oc describe quota core-object-counts -n demoproject
Name: core-object-counts
Namespace: demoproject
Resource Used Hard

configmaps 3 10
persistentvolumeclaims 0 4
replicationcontrollers 3 20
secrets 9 10
services 2 10
```

### 8.2.7. Configuring quota synchronization period

When a set of resources are deleted, the synchronization time frame of resources is determined by the **resource-quota-sync-period** setting in the `/etc/origin/master/master-config.yaml` file.

Before quota usage is restored, a user can encounter problems when attempting to reuse the resources. You can change the **resource-quota-sync-period** setting to have the set of resources regenerate in the needed amount of time (in seconds) for the resources to be once again available:

#### Example resource-quota-sync-period setting

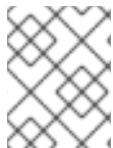
```
kubernetesMasterConfig:
apiLevels:
- v1beta3
- v1
apiServerArguments: null
```

```
controllerArguments:
 resource-quota-sync-period:
 - "10s"
```

After making any changes, restart the controller services to apply them.

```
$ master-restart api
$ master-restart controllers
```

Adjusting the regeneration time can be helpful for creating resources and determining resource usage when automation is used.



#### NOTE

The **resource-quota-sync-period** setting balances system performance. Reducing the sync period can result in a heavy load on the controller.

### 8.2.8. Explicit quota to consume a resource

If a resource is not managed by quota, a user has no restriction on the amount of resource that can be consumed. For example, if there is no quota on storage related to the gold storage class, the amount of gold storage a project can create is unbounded.

For high-cost compute or storage resources, administrators can require an explicit quota be granted to consume a resource. For example, if a project was not explicitly given quota for storage related to the gold storage class, users of that project would not be able to create any storage of that type.

In order to require explicit quota to consume a particular resource, the following stanza should be added to the master-config.yaml.

```
admissionConfig:
 pluginConfig:
 ResourceQuota:
 configuration:
 apiVersion: resourcequota.admission.k8s.io/v1alpha1
 kind: Configuration
 limitedResources:
 - resource: persistentvolumeclaims ①
 matchContains:
 - gold.storageclass.storage.k8s.io/requests.storage ②
```

**①** The group or resource to whose consumption is limited by default.

**②** The name of the resource tracked by quota associated with the group/resource to limit by default.

In the above example, the quota system intercepts every operation that creates or updates a **PersistentVolumeClaim**. It checks what resources controlled by quota would be consumed. If there is no covering quota for those resources in the project, the request is denied. In this example, if a user creates a **PersistentVolumeClaim** that uses storage associated with the gold storage class and there is no matching quota in the project, the request is denied.

### Additional resources

For examples of how to create the file needed to set quotas, see [Resources managed by quotas](#).

A description of how to allocate [compute resources managed by quota](#).

For information on managing limits and quota on project resources, see [Working with projects](#).

If a quota has been defined for your project, see [Understanding deployments](#) for considerations in cluster configurations.

## 8.3. SETTING LIMIT RANGES

A limit range, defined by a **LimitRange** object, defines compute resource constraints at the pod, container, image, image stream, and persistent volume claim level. The limit range specifies the amount of resources that a pod, container, image, image stream, or persistent volume claim can consume.

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. If the resource does not set an explicit value, and if the constraint supports a default value, the default value is applied to the resource.

For CPU and memory limits, if you specify a maximum value but do not specify a minimum limit, the resource can consume more CPU and memory resources than the maximum value.

### Core limit range object definition

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
 name: "core-resource-limits" 1
spec:
 limits:
 - type: "Pod"
 max:
 cpu: "2" 2
 memory: "1Gi" 3
 min:
 cpu: "200m" 4
 memory: "6Mi" 5
 - type: "Container"
 max:
 cpu: "2" 6
 memory: "1Gi" 7
 min:
 cpu: "100m" 8
 memory: "4Mi" 9
 default:
 cpu: "300m" 10
 memory: "200Mi" 11
 defaultRequest:
 cpu: "200m" 12
 memory: "100Mi" 13
 maxLimitRequestRatio:
 cpu: "10" 14
```

**1** The name of the limit range object.

- 2 The maximum amount of CPU that a pod can request on a node across all containers.
- 3 The maximum amount of memory that a pod can request on a node across all containers.
- 4 The minimum amount of CPU that a pod can request on a node across all containers. If you do not set a **min** value or you set **min** to **0**, the result is no limit and the pod can consume more than the **max** CPU value.
- 5 The minimum amount of memory that a pod can request on a node across all containers. If you do not set a **min** value or you set **min** to **0**, the result is no limit and the pod can consume more than the **max** memory value.
- 6 The maximum amount of CPU that a single container in a pod can request.
- 7 The maximum amount of memory that a single container in a pod can request.
- 8 The minimum amount of CPU that a single container in a pod can request. If you do not set a **min** value or you set **min** to **0**, the result is no limit and the pod can consume more than the **max** CPU value.
- 9 The minimum amount of memory that a single container in a pod can request. If you do not set a **min** value or you set **min** to **0**, the result is no limit and the pod can consume more than the **max** memory value.
- 10 The default CPU limit for a container if you do not specify a limit in the pod specification.
- 11 The default memory limit for a container if you do not specify a limit in the pod specification.
- 12 The default CPU request for a container if you do not specify a request in the pod specification.
- 13 The default memory request for a container if you do not specify a request in the pod specification.
- 14 The maximum limit-to-request ratio for a container.

## OpenShift Container Platform Limit range object definition

```

apiVersion: "v1"
kind: "LimitRange"
metadata:
 name: "openshift-resource-limits"
spec:
 limits:
 - type: openshift.io/Image
 max:
 storage: 1Gi ①
 - type: openshift.io/ImageStream
 max:
 openshift.io/image-tags: 20 ②
 openshift.io/images: 30 ③
 - type: "Pod"
 max:
 cpu: "2" ④
 memory: "1Gi" ⑤
 ephemeral-storage: "1Gi" ⑥

```

```
min:
 cpu: "1" ⑦
 memory: "1Gi" ⑧
```

- 1 The maximum size of an image that can be pushed to an internal registry.
- 2 The maximum number of unique image tags as defined in the specification for the image stream.
- 3 The maximum number of unique image references as defined in the specification for the image stream status.
- 4 The maximum amount of CPU that a pod can request on a node across all containers.
- 5 The maximum amount of memory that a pod can request on a node across all containers.
- 6 The maximum amount of ephemeral storage that a pod can request on a node across all containers.
- 7 The minimum amount of CPU that a pod can request on a node across all containers. See the Supported Constraints table for important information.
- 8 The minimum amount of memory that a pod can request on a node across all containers. If you do not set a **min** value or you set **min** to **0**, the result is no limit and the pod can consume more than the **max** memory value.

You can specify both core and OpenShift Container Platform resources in one limit range object.

### 8.3.1. Container limits

#### Supported Resources:

- CPU
- Memory

#### Supported Constraints

Per container, the following must hold true if specified:

#### Container

| Constraint | Behavior                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Min        | <p><b>Min[&lt;resource&gt;]</b> less than or equal to<br/> <b>container.resources.requests[&lt;resource&gt;]</b> (required) less than or equal to <b>container/resources.limits[&lt;resource&gt;]</b> (optional)</p> <p>If the configuration defines a <b>min</b> CPU, the request value must be greater than the CPU value. If you do not set a <b>min</b> value or you set <b>min</b> to <b>0</b>, the result is no limit and the pod can consume more of the resource than the <b>max</b> value.</p> |

| Constraint                  | Behavior                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Max</b>                  | <p><b>container.resources.limits[&lt;resource&gt;]</b> (required) less than or equal to <b>Max[&lt;resource&gt;]</b></p> <p>If the configuration defines a <b>max</b> CPU, you do not need to define a CPU request value. However, you must set a limit that satisfies the maximum CPU constraint that is specified in the limit range.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>MaxLimitRequestRatio</b> | <p><b>MaxLimitRequestRatio[&lt;resource&gt;]</b> less than or equal to <b>(container.resources.limits[&lt;resource&gt;] / container.resources.requests[&lt;resource&gt;])</b></p> <p>If the limit range defines a <b>maxLimitRequestRatio</b> constraint, any new containers must have both a <b>request</b> and a <b>limit</b> value. Additionally, OpenShift Container Platform calculates a limit-to-request ratio by dividing the <b>limit</b> by the <b>request</b>. The result should be an integer greater than 1.</p> <p>For example, if a container has <b>cpu: 500</b> in the <b>limit</b> value, and <b>cpu: 100</b> in the <b>request</b> value, the limit-to-request ratio for <b>cpu</b> is <b>5</b>. This ratio must be less than or equal to the <b>maxLimitRequestRatio</b>.</p> |

#### Supported Defaults:

##### **Default[<resource>]**

Defaults **container.resources.limit[<resource>]** to specified value if none.

##### **Default Requests[<resource>]**

Defaults **container.resources.requests[<resource>]** to specified value if none.

### 8.3.2. Pod limits

#### Supported Resources:

- CPU
- Memory

#### Supported Constraints:

Across all containers in a pod, the following must hold true:

**Table 8.4. Pod**

| Constraint | Enforced Behavior |
|------------|-------------------|
|------------|-------------------|

| Constraint                  | Enforced Behavior                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Min</b>                  | <b>Min[&lt;resource&gt;]</b> less than or equal to <b>container.resources.requests[&lt;resource&gt;]</b> (required) less than or equal to <b>container.resources.limits[&lt;resource&gt;]</b> . If you do not set a <b>min</b> value or you set <b>min</b> to <b>0</b> , the result is no limit and the pod can consume more of the resource than the <b>max</b> value. |
| <b>Max</b>                  | <b>container.resources.limits[&lt;resource&gt;]</b> (required) less than or equal to <b>Max[&lt;resource&gt;]</b> .                                                                                                                                                                                                                                                     |
| <b>MaxLimitRequestRatio</b> | <b>MaxLimitRequestRatio[&lt;resource&gt;]</b> less than or equal to <b>(container.resources.limits[&lt;resource&gt;] / container.resources.requests[&lt;resource&gt;])</b> .                                                                                                                                                                                            |

### 8.3.3. Image limits

Supported Resources:

- Storage

Resource type name:

- **openshift.io/Image**

Per image, the following must hold true if specified:

Table 8.5. Image

| Constraint | Behavior                                                                                 |
|------------|------------------------------------------------------------------------------------------|
| <b>Max</b> | <b>image.dockerimagemetadata.size</b> less than or equal to <b>Max[&lt;resource&gt;]</b> |



#### NOTE

To prevent blobs that exceed the limit from being uploaded to the registry, the registry must be configured to enforce quota. The **REGISTRY\_MIDDLEWARE\_REPOSITORY\_OPENSHIFT\_ENFORCEQUOTA** environment variable must be set to **true**. By default, the environment variable is set to **true** for new deployments.

### 8.3.4. Image stream limits

Supported Resources:

- **openshift.io/image-tags**
- **openshift.io/images**

Resource type name:

- **openshift.io/ImageStream**

Per image stream, the following must hold true if specified:

**Table 8.6. ImageStream**

| Constraint                          | Behavior                                                                                                                                                                                                                              |
|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Max[openshift.io/image-tags]</b> | <b>length( uniqueimagetags( imagestream.spec.tags ) )</b> less than or equal to <b>Max[openshift.io/image-tags]</b><br><b>uniqueimagetags</b> returns unique references to images of given spec tags.                                 |
| <b>Max[openshift.io/images]</b>     | <b>length( uniqueimages( imagestream.status.tags ) )</b> less than or equal to <b>Max[openshift.io/images]</b><br><b>uniqueimages</b> returns unique image names found in status tags. The name is equal to the digest for the image. |

### 8.3.5. Counting of image references

The **openshift.io/image-tags** resource represents unique stream limits. Possible references are an **ImageStreamTag**, an **ImageStreamImage**, or a **DockerImage**. Tags can be created by using the **oc tag** and **oc import-image** commands or by using image streams. No distinction is made between internal and external references. However, each unique reference that is tagged in an image stream 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 that are recorded in image stream status. It helps to restrict several images that can be pushed to the internal registry. Internal and external references are not distinguished.

### 8.3.6. PersistentVolumeClaim limits

**Supported Resources:**

- Storage

**Supported Constraints:**

Across all persistent volume claims in a project, the following must hold true:

**Table 8.7. Pod**

| Constraint | Enforced Behavior                                                                                                |
|------------|------------------------------------------------------------------------------------------------------------------|
| <b>Min</b> | <code>Min[&lt;resource&gt;] &lt;= claim.spec.resources.requests[&lt;resource&gt;]</code> (required)              |
| <b>Max</b> | <code>claim.spec.resources.requests[&lt;resource&gt;]</code> (required) <code>&lt;= Max[&lt;resource&gt;]</code> |

### Limit Range Object Definition

```
{
 "apiVersion": "v1",
 "kind": "LimitRange",
 "metadata": {
 "name": "pvcs" ①
 },
 "spec": {
 "limits": [
 {
 "type": "PersistentVolumeClaim",
 "min": {
 "storage": "2Gi" ②
 },
 "max": {
 "storage": "50Gi" ③
 }
 }
]
 }
}
```

- ① The name of the limit range object.
- ② The minimum amount of storage that can be requested in a persistent volume claim.
- ③ The maximum amount of storage that can be requested in a persistent volume claim.

## Additional resources

For information on stream limits, see [managing images streams](#).

For information on [stream limits](#).

For more information on [compute resource constraints](#).

For more information on how CPU and memory are measured, see [Recommended control plane practices](#).

You can specify limits and requests for ephemeral storage. For more information on this feature, see [Understanding ephemeral storage](#).

## 8.4. LIMIT RANGE OPERATIONS

### 8.4.1. Creating a limit range

Shown here is an example procedure to follow for creating a limit range.

#### Procedure

1. Create the object:

```
$ oc create -f <limit_range_file> -n <project>
```

### 8.4.2. View the limit

You can view any limit ranges that are defined in a project by navigating in the web console to the **Quota** page for the project. You can also use the CLI to view limit range details by performing the following steps:

### Procedure

1. Get the list of limit range objects that are defined in the project. For example, a project called **demoproject**:

```
$ oc get limits -n demoproject
```

### Example Output

| NAME            | AGE |
|-----------------|-----|
| resource-limits | 6d  |

2. Describe the limit range. For example, for a limit range called **resource-limits**:

```
$ oc describe limits resource-limits -n demoproject
```

### Example Output

| Name:                    | resource-limits         | Namespace: | demoproject | Type                | Resource | Min  | Max | Default Request | Default Limit | Max |
|--------------------------|-------------------------|------------|-------------|---------------------|----------|------|-----|-----------------|---------------|-----|
| Pod                      | cpu                     |            |             | Limit/Request Ratio |          | 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  | -               | -             | -   |

### 8.4.3. Deleting a limit range

To remove a limit range, run the following command:

+

```
$ oc delete limits <limit_name>
```

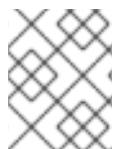
S

### Additional resources

For information about enforcing different limits on the number of projects that your users can create, managing limits, and quota on project resources, see [Resource quotas per projects](#).

# CHAPTER 9. RECOMMENDED HOST PRACTICES FOR IBM Z & IBM LINUXONE ENVIRONMENTS

This topic provides recommended host practices for OpenShift Container Platform on IBM Z® and IBM® LinuxONE.



## NOTE

The s390x architecture is unique in many aspects. Therefore, some recommendations made here might not apply to other platforms.



## NOTE

Unless stated otherwise, these practices apply to both z/VM and Red Hat Enterprise Linux (RHEL) KVM installations on IBM Z® and IBM® LinuxONE.

## 9.1. MANAGING CPU OVERCOMMITMENT

In a highly virtualized IBM Z® environment, you must carefully plan the infrastructure setup and sizing. One of the most important features of virtualization is the capability to do resource overcommitment, allocating more resources to the virtual machines than actually available at the hypervisor level. This is very workload dependent and there is no golden rule that can be applied to all setups.

Depending on your setup, consider these best practices regarding CPU overcommitment:

- At LPAR level (PR/SM hypervisor), avoid assigning all available physical cores (IFLs) to each LPAR. For example, with four physical IFLs available, you should not define three LPARs with four logical IFLs each.
- Check and understand LPAR shares and weights.
- An excessive number of virtual CPUs can adversely affect performance. Do not define more virtual processors to a guest than logical processors are defined to the LPAR.
- Configure the number of virtual processors per guest for peak workload, not more.
- Start small and monitor the workload. Increase the vCPU number incrementally if necessary.
- Not all workloads are suitable for high overcommitment ratios. If the workload is CPU intensive, you will probably not be able to achieve high ratios without performance problems. Workloads that are more I/O intensive can keep consistent performance even with high overcommitment ratios.

### Additional resources

- [z/VM Common Performance Problems and Solutions](#)
- [z/VM overcommitment considerations](#)
- [LPAR CPU management](#)

## 9.2. DISABLE TRANSPARENT HUGE PAGES

Transparent Huge Pages (THP) attempt to automate most aspects of creating, managing, and using

huge pages. Since THP automatically manages the huge pages, this is not always handled optimally for all types of workloads. THP can lead to performance regressions, since many applications handle huge pages on their own. Therefore, consider disabling THP.

## 9.3. BOOST NETWORKING PERFORMANCE WITH RECEIVE FLOW STEERING

Receive Flow Steering (RFS) extends Receive Packet Steering (RPS) by further reducing network latency. RFS is technically based on RPS, and improves the efficiency of packet processing by increasing the CPU cache hit rate. RFS achieves this, and in addition considers queue length, by determining the most convenient CPU for computation so that cache hits are more likely to occur within the CPU. Thus, the CPU cache is invalidated less and requires fewer cycles to rebuild the cache. This can help reduce packet processing run time.

### 9.3.1. Use the Machine Config Operator (MCO) to activate RFS

#### Procedure

1. Copy the following MCO sample profile into a YAML file. For example, `enable-rfs.yaml`:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
 labels:
 machineconfiguration.openshift.io/role: worker
 name: 50-enable-rfs
spec:
 config:
 ignition:
 version: 2.2.0
 storage:
 files:
 - contents:
 source: data:text/plain;charset=US-
ASCII,%23%20turn%20on%20Receive%20Flow%20Steering%20%28RFS%29%20for%20all
%20network%20interfaces%0ASUBSYSTEM%3D%3D%22net%22%2C%20ACTION%3D%
3D%22add%22%2C%20RUN%7Bprogram%7D%2B%3D%22/bin/bash%20-
c%20%27for%20x%20in%20/sys/%24DEVPATH/queues/rx-
%2A%3B%20do%20echo%208192%20%3E%20%24x/rps_flow_cnt%3B%20%20done%27
%22%0A
 filesystem: root
 mode: 0644
 path: /etc/udev/rules.d/70-persistent-net.rules
 - contents:
 source: data:text/plain;charset=US-
ASCII,%23%20define%20sock%20flow%20enbtried%20for%20%20Receive%20Flow%20Ste
ering%20%28RFS%29%0Anet.core.rps_sock_flow_entries%3D8192%0A
 filesystem: root
 mode: 0644
 path: /etc/sysctl.d/95-enable-rps.conf
```

2. Create the MCO profile:

```
$ oc create -f enable-rfs.yaml
```

3. Verify that an entry named **50-enable-rfs** is listed:

```
$ oc get mc
```

4. To deactivate, enter:

```
$ oc delete mc 50-enable-rfs
```

#### Additional resources

- [OpenShift Container Platform on IBM Z®: Tune your network performance with RFS](#)
- [Configuring Receive Flow Steering \(RFS\)](#)
- [Scaling in the Linux Networking Stack](#)

## 9.4. CHOOSE YOUR NETWORKING SETUP

The networking stack is one of the most important components for a Kubernetes-based product like OpenShift Container Platform. For IBM Z® setups, the networking setup depends on the hypervisor of your choice. Depending on the workload and the application, the best fit usually changes with the use case and the traffic pattern.

Depending on your setup, consider these best practices:

- Consider all options regarding networking devices to optimize your traffic pattern. Explore the advantages of OSA-Express, RoCE Express, HiperSockets, z/VM VSwitch, Linux Bridge (KVM), and others to decide which option leads to the greatest benefit for your setup.
- Always use the latest available NIC version. For example, OSA Express 7S 10 GbE shows great improvement compared to OSA Express 6S 10 GbE with transactional workload types, although both are 10 GbE adapters.
- Each virtual switch adds an additional layer of latency.
- The load balancer plays an important role for network communication outside the cluster. Consider using a production-grade hardware load balancer if this is critical for your application.
- OpenShift Container Platform OVN-Kubernetes network plugin introduces flows and rules, which impact the networking performance. Make sure to consider pod affinities and placements, to benefit from the locality of services where communication is critical.
- Balance the trade-off between performance and functionality.

#### Additional resources

- [OpenShift Container Platform on IBM Z® - Performance Experiences, Hints and Tips](#)
- [OpenShift Container Platform on IBM Z® Networking Performance](#)
- [Controlling pod placement on nodes using node affinity rules](#)

## 9.5. ENSURE HIGH DISK PERFORMANCE WITH HYPERPAV ON Z/VM

DASD and ECKD devices are commonly used disk types in IBM Z® environments. In a typical OpenShift Container Platform setup in z/VM environments, DASD disks are commonly used to support the local storage for the nodes. You can set up HyperPAV alias devices to provide more throughput and overall better I/O performance for the DASD disks that support the z/VM guests.

Using HyperPAV for the local storage devices leads to a significant performance benefit. However, you must be aware that there is a trade-off between throughput and CPU costs.

### 9.5.1. Use the Machine Config Operator (MCO) to activate HyperPAV aliases in nodes using z/VM full-pack minidisks

For z/VM-based OpenShift Container Platform setups that use full-pack minidisks, you can leverage the advantage of MCO profiles by activating HyperPAV aliases in all of the nodes. You must add YAML configurations for both control plane and compute nodes.

#### Procedure

1. Copy the following MCO sample profile into a YAML file for the control plane node. For example, **05-master-kernelarg-hpav.yaml**:

```
$ cat 05-master-kernelarg-hpav.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
 labels:
 machineconfiguration.openshift.io/role: master
 name: 05-master-kernelarg-hpav
spec:
 config:
 ignition:
 version: 3.1.0
 kernelArguments:
 - rd.dasd=800-805
```

2. Copy the following MCO sample profile into a YAML file for the compute node. For example, **05-worker-kernelarg-hpav.yaml**:

```
$ cat 05-worker-kernelarg-hpav.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
 labels:
 machineconfiguration.openshift.io/role: worker
 name: 05-worker-kernelarg-hpav
spec:
 config:
 ignition:
 version: 3.1.0
 kernelArguments:
 - rd.dasd=800-805
```



#### NOTE

You must modify the **rd.dasd** arguments to fit the device IDs.

3. Create the MCO profiles:

```
$ oc create -f 05-master-kernelarg-hpav.yaml
$ oc create -f 05-worker-kernelarg-hpav.yaml
```

4. To deactivate, enter:

```
$ oc delete -f 05-master-kernelarg-hpav.yaml
$ oc delete -f 05-worker-kernelarg-hpav.yaml
```

#### Additional resources

- [Using HyperPAV for ECKD DASD](#)
- [Scaling HyperPAV alias devices on Linux guests on z/VM](#)

## 9.6. RHEL KVM ON IBM Z HOST RECOMMENDATIONS

Optimizing a KVM virtual server environment strongly depends on the workloads of the virtual servers and on the available resources. The same action that enhances performance in one environment can have adverse effects in another. Finding the best balance for a particular setting can be a challenge and often involves experimentation.

The following section introduces some best practices when using OpenShift Container Platform with RHEL KVM on IBM Z® and IBM® LinuxONE environments.

### 9.6.1. Use I/O threads for your virtual block devices

To make virtual block devices use I/O threads, you must configure one or more I/O threads for the virtual server and each virtual block device to use one of these I/O threads.

The following example specifies `<iotreads>3</iotreads>` to configure three I/O threads, with consecutive decimal thread IDs 1, 2, and 3. The `iothread="2"` parameter specifies the driver element of the disk device to use the I/O thread with ID 2.

#### Sample I/O thread specification

```
...
<domain>
<iotreads>3</iotreads>①
...
<devices>
...
<disk type="block" device="disk">②
<driver ... iothread="2"/>
</disk>
...
</devices>
...
</domain>
```

- 1 The number of I/O threads.
- 2 The driver element of the disk device.

Threads can increase the performance of I/O operations for disk devices, but they also use memory and CPU resources. You can configure multiple devices to use the same thread. The best mapping of threads to devices depends on the available resources and the workload.

Start with a small number of I/O threads. Often, a single I/O thread for all disk devices is sufficient. Do not configure more threads than the number of virtual CPUs, and do not configure idle threads.

You can use the **virsh iothreadadd** command to add I/O threads with specific thread IDs to a running virtual server.

### 9.6.2. Avoid virtual SCSI devices

Configure virtual SCSI devices only if you need to address the device through SCSI-specific interfaces. Configure disk space as virtual block devices rather than virtual SCSI devices, regardless of the backing on the host.

However, you might need SCSI-specific interfaces for:

- A LUN for a SCSI-attached tape drive on the host.
- A DVD ISO file on the host file system that is mounted on a virtual DVD drive.

### 9.6.3. Configure guest caching for disk

Configure your disk devices to do caching by the guest and not by the host.

Ensure that the driver element of the disk device includes the **cache="none"** and **io="native"** parameters.

```
<disk type="block" device="disk">
 <driver name="qemu" type="raw" cache="none" io="native" iothread="1"/>
 ...
</disk>
```

### 9.6.4. Exclude the memory balloon device

Unless you need a dynamic memory size, do not define a memory balloon device and ensure that libvirt does not create one for you. Include the **memballoon** parameter as a child of the devices element in your domain configuration XML file.

- Check the list of active profiles:

```
 <memballoon model="none"/>
```

### 9.6.5. Tune the CPU migration algorithm of the host scheduler



## IMPORTANT

Do not change the scheduler settings unless you are an expert who understands the implications. Do not apply changes to production systems without testing them and confirming that they have the intended effect.

The **kernel.sched\_migration\_cost\_ns** parameter specifies a time interval in nanoseconds. After the last execution of a task, the CPU cache is considered to have useful content until this interval expires. Increasing this interval results in fewer task migrations. The default value is 500000 ns.

If the CPU idle time is higher than expected when there are runnable processes, try reducing this interval. If tasks bounce between CPUs or nodes too often, try increasing it.

To dynamically set the interval to 60000 ns, enter the following command:

```
sysctl kernel.sched_migration_cost_ns=60000
```

To persistently change the value to 60000 ns, add the following entry to **/etc/sysctl.conf**:

```
kernel.sched_migration_cost_ns=60000
```

### 9.6.6. Disable the cpuset cgroup controller



#### NOTE

This setting applies only to KVM hosts with cgroups version 1. To enable CPU hotplug on the host, disable the cgroup controller.

#### Procedure

1. Open **/etc/libvirt/qemu.conf** with an editor of your choice.
2. Go to the **cgroup\_controllers** line.
3. Duplicate the entire line and remove the leading number sign (#) from the copy.
4. Remove the **cpuset** entry, as follows:

```
cgroup_controllers = ["cpu", "devices", "memory", "blkio", "cpuacct"]
```

5. For the new setting to take effect, you must restart the libvirtd daemon:

- a. Stop all virtual machines.

- b. Run the following command:

```
systemctl restart libvirdt
```

- c. Restart the virtual machines.

This setting persists across host reboots.

### 9.6.7. Tune the polling period for idle virtual CPUs

When a virtual CPU becomes idle, KVM polls for wakeup conditions for the virtual CPU before allocating the host resource. You can specify the time interval, during which polling takes place in sysfs at **/sys/module/kvm/parameters/halt\_poll\_ns**. During the specified time, polling reduces the wakeup latency for the virtual CPU at the expense of resource usage. Depending on the workload, a longer or shorter time for polling can be beneficial. The time interval is specified in nanoseconds. The default is 50000 ns.

- To optimize for low CPU consumption, enter a small value or write 0 to disable polling:

```
echo 0 > /sys/module/kvm/parameters/halt_poll_ns
```

- To optimize for low latency, for example for transactional workloads, enter a large value:

```
echo 80000 > /sys/module/kvm/parameters/halt_poll_ns
```

## Additional resources

- [Linux on IBM Z® Performance Tuning for KVM](#)
- [Getting started with virtualization on IBM Z®](#)

# CHAPTER 10. 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.

## 10.1. ABOUT THE NODE TUNING OPERATOR

The Node Tuning Operator helps you manage node-level tuning by orchestrating the TuneD daemon and achieves low latency performance by using the Performance Profile controller. The majority of high-performance applications require some level of kernel tuning. The Node Tuning Operator provides a unified management interface to users of node-level sysctls and more flexibility to add custom tuning specified by user needs.

The Operator manages the containerized TuneD daemon for OpenShift Container Platform as a Kubernetes daemon set. It ensures the custom tuning specification is passed to all containerized TuneD daemons running in the cluster in the format that the daemons understand. The daemons run on all nodes in the cluster, one per node.

Node-level settings applied by the containerized TuneD daemon are rolled back on an event that triggers a profile change or when the containerized TuneD daemon is terminated gracefully by receiving and handling a termination signal.

The Node Tuning Operator uses the Performance Profile controller to implement automatic tuning to achieve low latency performance for OpenShift Container Platform applications.

The cluster administrator configures a performance profile to define node-level settings such as the following:

- Updating the kernel to kernel-rt.
- Choosing CPUs for housekeeping.
- Choosing CPUs for running workloads.

The Node Tuning Operator is part of a standard OpenShift Container Platform installation in version 4.1 and later.



### NOTE

In earlier versions of OpenShift Container Platform, the Performance Addon Operator was used to implement automatic tuning to achieve low latency performance for OpenShift applications. In OpenShift Container Platform 4.11 and later, this functionality is part of the Node Tuning Operator.

## 10.2. 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.

## 10.3. 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 {} \;
```

## 10.4. VERIFYING THAT THE TUNED PROFILES ARE APPLIED

Verify the TuneD profiles that are applied to your cluster node.

```
$ oc get profile.tuned.openshift.io -n openshift-cluster-node-tuning-operator
```

### Example output

NAME	TUNED	APPLIED	DEGRADED	AGE
master-0	openshift-control-plane	True	False	6h33m
master-1	openshift-control-plane	True	False	6h33m
master-2	openshift-control-plane	True	False	6h33m
worker-a	openshift-node	True	False	6h28m
worker-b	openshift-node	True	False	6h28m

- **NAME:** Name of the Profile object. There is one Profile object per node and their names match.
- **TUNED:** Name of the desired TuneD profile to apply.
- **APPLIED:** **True** if the TuneD daemon applied the desired profile. ( **True/False/Unknown**).
- **DEGRADED:** **True** if any errors were reported during application of the TuneD profile ( **True/False/Unknown**).
- **AGE:** Time elapsed since the creation of Profile object.

The **ClusterOperator/node-tuning** object also contains useful information about the Operator and its node agents' health. For example, Operator misconfiguration is reported by **ClusterOperator/node-tuning** status messages.

To get status information about the **ClusterOperator/node-tuning** object, run the following command:

```
$ oc get co/node-tuning -n openshift-cluster-node-tuning-operator
```

### Example output

NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE	MESSAGE
node-tuning	4.18.1	True	False	True	60m	1/5 Profiles with bootcmdline conflict

If either the **ClusterOperator/node-tuning** or a profile object's status is **DEGRADED**, additional information is provided in the Operator or operand logs.

## 10.5. 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: ①
 <mcLabels> ②
 match: ③
 <match> ④
 priority: <priority> ⑤
 profile: <tuned_profile_name> ⑥
 operand: ⑦
 debug: <bool> ⑧
 tunedConfig:
 reapply_sysctl: <bool> ⑨
```

① Optional.

- 2 A dictionary of key/value **MachineConfig** labels. The keys must be unique.
- 3 If omitted, profile match is assumed unless a profile with a higher priority matches first or **machineConfigLabels** is set.
- 4 An optional list.
- 5 Profile ordering priority. Lower numbers mean higher priority (**0** is the highest priority).
- 6 A TuneD profile to apply on a match. For example **tuned\_profile\_1**.
- 7 Optional operand configuration.
- 8 Turn debugging on or off for the TuneD daemon. Options are **true** for on or **false** for off. The default is **false**.
- 9 Turn **reapply\_sysctl** functionality on or off for the TuneD daemon. Options are **true** for on and **false** for off.

**<match>** is an optional list recursively defined as follows:

```
- label: <label_name> 1
 value: <label_value> 2
 type: <label_type> 3
 <match> 4
```

- 1 Node or pod label name.
- 2 Optional node or pod label value. If omitted, the presence of **<label\_name>** is enough to match.
- 3 Optional object type (**node** or **pod**). If omitted, **node** is assumed.
- 4 An optional **<match>** list.

If **<match>** is not omitted, all nested **<match>** sections must also evaluate to **true**. Otherwise, **false** is assumed and the profile with the respective **<match>** section will not be applied or recommended. Therefore, the nesting (child **<match>** sections) works as logical AND operator. Conversely, if any item of the **<match>** list matches, the entire **<match>** list evaluates to **true**. Therefore, the list acts as logical OR operator.

If **machineConfigLabels** is defined, machine config pool based matching is turned on for the given **recommend**: list item. **<mcLabels>** specifies the labels for a machine config. The machine config is created automatically to apply host settings, such as kernel boot parameters, for the profile **<tuned\_profile\_name>**. This involves finding all machine config pools with machine config selector matching **<mcLabels>** and setting the profile **<tuned\_profile\_name>** on all nodes that are assigned the found machine config pools. To target nodes that have both master and worker roles, you must use the master role.

The list items **match** and **machineConfigLabels** are connected by the logical OR operator. The **match** item is evaluated first in a short-circuit manner. Therefore, if it evaluates to **true**, the **machineConfigLabels** item is not considered.



## IMPORTANT

When using machine config pool based matching, it is advised to group nodes with the same hardware configuration into the same machine config pool. Not following this practice might result in TuneD operands calculating conflicting kernel parameters for two or more nodes sharing the same machine config pool.

### Example: Node or pod label based matching

```

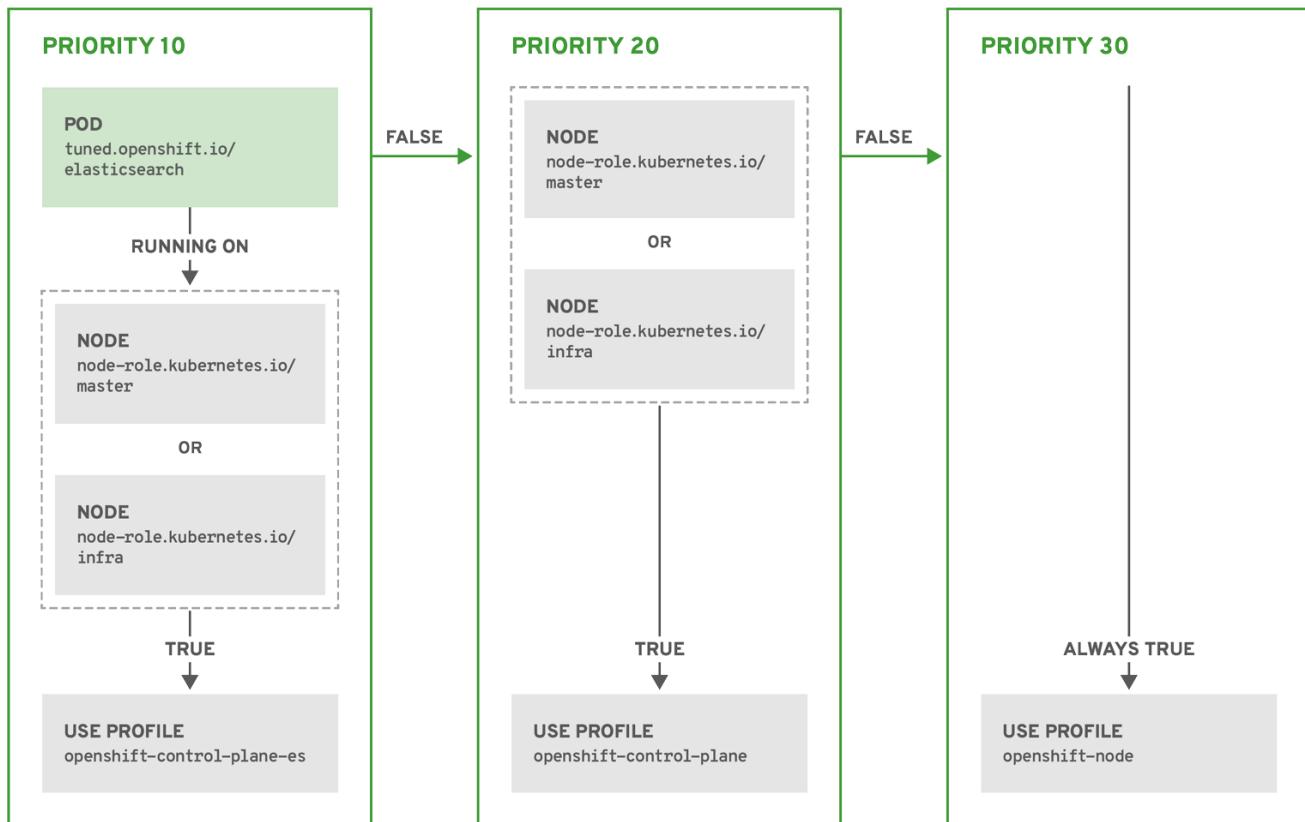
- match:
- label: tuned.openshift.io/elasticsearch
 match:
 - label: node-role.kubernetes.io/master
 - label: node-role.kubernetes.io/infra
 type: pod
 priority: 10
 profile: openshift-control-plane-es
- match:
 - label: node-role.kubernetes.io/master
 - label: node-role.kubernetes.io/infra
 priority: 20
 profile: openshift-control-plane
- priority: 30
 profile: openshift-node

```

The CR above is translated for the containerized TuneD daemon into its **recommend.conf** file based on the profile priorities. The profile with the highest priority (**10**) is **openshift-control-plane-es** and, therefore, it is considered first. The containerized TuneD daemon running on a given node looks to see if there is a pod running on the same node with the **tuned.openshift.io/elasticsearch** label set. If not, the entire **<match>** section evaluates as **false**. If there is such a pod with the label, in order for the **<match>** section to evaluate to **true**, the node label also needs to be **node-role.kubernetes.io/master** or **node-role.kubernetes.io/infra**.

If the labels for the profile with priority **10** matched, **openshift-control-plane-es** profile is applied and no other profile is considered. If the node/pod label combination did not match, the second highest priority profile (**openshift-control-plane**) is considered. This profile is applied if the containerized TuneD pod runs on a node with labels **node-role.kubernetes.io/master** or **node-role.kubernetes.io/infra**.

Finally, the profile **openshift-node** has the lowest priority of **30**. It lacks the **<match>** section and, therefore, will always match. It acts as a profile catch-all to set **openshift-node** profile, if no other profile with higher priority matches on a given node.



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.

## 10.6. CUSTOM TUNING EXAMPLES

### Using TuneD profiles from the default CR

The following CR applies custom node-level tuning for OpenShift Container Platform nodes with label **tuned.openshift.io/ingress-node-label** set to any value.

#### Example: custom tuning using the **openshift-control-plane** TuneD profile

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
 name: ingress
 namespace: openshift-cluster-node-tuning-operator
spec:
 profile:
 - data: |
 [main]
 summary=A custom OpenShift ingress profile
 include=openshift-control-plane
```

```
[sysctl]
net.ipv4.ip_local_port_range="1024 65535"
net.ipv4.tcp_tw_reuse=1
name: openshift-ingress
recommend:
- match:
 - label: tuned.openshift.io/ingress-node-label
priority: 10
profile: openshift-ingress
```



## IMPORTANT

Custom profile writers are strongly encouraged to include the default TuneD daemon profiles shipped within the default Tuned CR. The example above uses the default **openshift-control-plane** profile to accomplish this.

### Using built-in TuneD profiles

Given the successful rollout of the NTO-managed daemon set, the TuneD operands all manage the same version of the TuneD daemon. To list the built-in TuneD profiles supported by the daemon, query any TuneD pod in the following way:

```
$ oc exec $tuned_pod -n openshift-cluster-node-tuning-operator -- find /usr/lib/tuned/ -name tuned.conf -printf '%h\n' | sed 's|^.*|'|'
```

You can use the profile names retrieved by this in your custom tuning specification.

### Example: using built-in hpc-compute TuneD profile

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
 name: openshift-node-hpc-compute
 namespace: openshift-cluster-node-tuning-operator
spec:
 profile:
 - data: |
 [main]
 summary=Custom OpenShift node profile for HPC compute workloads
 include=openshift-node,hpc-compute
 name: openshift-node-hpc-compute

 recommend:
 - match:
 - label: tuned.openshift.io/openshift-node-hpc-compute
 priority: 20
 profile: openshift-node-hpc-compute
```

In addition to the built-in **hpc-compute** profile, the example above includes the **openshift-node** TuneD daemon profile shipped within the default Tuned CR to use OpenShift-specific tuning for compute nodes.

### Overriding host-level sysctls

Various kernel parameters can be changed at runtime by using `/run/sysctl.d/`, `/etc/sysctl.d/`, and `/etc/sysctl.conf` host configuration files. OpenShift Container Platform adds several host configuration files which set kernel parameters at runtime; for example, `net.ipv[4-6].`, `fs.inotify.`, and `vm.max_map_count`. These runtime parameters provide basic functional tuning for the system prior to the kubelet and the Operator start.

The Operator does not override these settings unless the `reapply_sysctl` option is set to `false`. Setting this option to `false` results in **TuneD** not applying the settings from the host configuration files after it applies its custom profile.

### Example: overriding host-level sysctls

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
 name: openshift-no-reapply-sysctl
 namespace: openshift-cluster-node-tuning-operator
spec:
 profile:
 - data: |
 [main]
 summary=Custom OpenShift profile
 include=openshift-node
 [sysctl]
 vm.max_map_count=>524288
 name: openshift-no-reapply-sysctl
 recommend:
 - match:
 - label: tuned.openshift.io/openshift-no-reapply-sysctl
 priority: 15
 profile: openshift-no-reapply-sysctl
 operand:
 tunedConfig:
 reapply_sysctl: false
```

## 10.7. DEFERRING APPLICATION OF TUNING CHANGES

As an administrator, use the Node Tuning Operator (NTO) to update custom resources (CRs) on a running system and make tuning changes. For example, they can update or add a sysctl parameter to the `[sysctl]` section of the tuned object. When administrators apply a tuning change, the NTO prompts TuneD to reprocess all configurations, causing the tuned process to roll back all tuning and then reapply it.

Latency-sensitive applications may not tolerate the removal and reapplication of the tuned profile, as it can briefly disrupt performance. This is particularly critical for configurations that partition CPUs and manage process or interrupt affinity using the performance profile. To avoid this issue, OpenShift Container Platform introduced new methods for applying tuning changes. Before OpenShift Container Platform 4.17, the only available method, immediate, applied changes instantly, often triggering a tuned restart.

The following additional methods are supported:

- **always**: Every change is applied at the next node restart.

- **update:** When a tuning change modifies a tuned profile, it is applied immediately by default and takes effect as soon as possible. When a tuning change does not cause a tuned profile to change and its values are modified in place, it is treated as always.

Enable this feature by adding the annotation **tuned.openshift.io/deferred**. The following table summarizes the possible values for the annotation:

Annotation value	Description
missing	The change is applied immediately.
always	The change is applied at the next node restart.
update	The change is applied immediately if it causes a profile change, otherwise at the next node restart.

The following example demonstrates how to apply a change to the **kernel.shmmni** sysctl parameter by using the **always** method:

## Example

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
 name: performance-patch
 namespace: openshift-cluster-node-tuning-operator
 annotations:
 tuned.openshift.io/deferred: "always"
spec:
 profile:
 - name: performance-patch
 data: |
 [main]
 summary=Configuration changes profile inherited from performance created tuned
 include=openshift-node-performance-performance ①
 [sysctl]
 kernel.shmmni=8192 ②
 recommend:
 - machineConfigLabels:
 machineconfiguration.openshift.io/role: worker-cnf ③
 priority: 19
 profile: performance-patch
```

- ① The **include** directive is used to inherit the **openshift-node-performance-performance** profile. This is a best practice to ensure that the profile is not missing any required settings.
- ② The **kernel.shmmni** sysctl parameter is being changed to **8192**.
- ③ The **machineConfigLabels** field is used to target the **worker-cnf** role. Configure a **MachineConfigPool** resource to ensure the profile is applied only to the correct nodes.

### 10.7.1. Deferring application of tuning changes: An example

The following worked example describes how to defer the application of tuning changes by using the Node Tuning Operator.

#### Prerequisites

- You have **cluster-admin** role access.
- You have applied a performance profile to your cluster.
- A **MachineConfigPool** resource, for example, **worker-cnf** is configured to ensure that the profile is only applied to the designated nodes.

#### Procedure

1. Check what profiles are currently applied to your cluster by running the following command:

```
$ oc -n openshift-cluster-node-tuning-operator get tuned
```

#### Example output

NAME	AGE
default	63m
openshift-node-performance-performance	21m

2. Check the machine config pools in your cluster by running the following command:

```
$ oc get mcp
```

#### Example output

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT		
DEGRADED MACHINECOUNT	AGE			
master	rendered-master-79a26af9f78ced61fa8cccd309d3c859c	True	False	
False	3 3 3 0 157m			
worker	rendered-worker-d9352e91a1b14de7ef453fa54480ce0e	True	False	
False	2 2 2 0 157m			
worker-cnf	rendered-worker-cnf-f398fc4fcb2b20104a51e744b8247272	True	False	
False	1 1 1 0 92m			

3. Describe the current applied performance profile by running the following command:

```
$ oc describe performanceprofile performance | grep Tuned
```

#### Example output

Tuned:	openshift-cluster-node-tuning-operator/openshift-node-performance-performance
--------	-------------------------------------------------------------------------------

4. Verify the existing value of the **kernel.shmmni** sysctl parameter:

- Run the following command to display the node names:

```
$ oc get nodes
```

#### Example output

NAME	STATUS	ROLES	AGE	VERSION
ip-10-0-26-151.ec2.internal	Ready	worker,worker-cnf	116m	v1.30.6
ip-10-0-46-60.ec2.internal	Ready	worker	115m	v1.30.6
ip-10-0-52-141.ec2.internal	Ready	control-plane,master	123m	v1.30.6
ip-10-0-6-97.ec2.internal	Ready	control-plane,master	121m	v1.30.6
ip-10-0-86-145.ec2.internal	Ready	worker	117m	v1.30.6
ip-10-0-92-228.ec2.internal	Ready	control-plane,master	123m	v1.30.6

- Run the following command to display the current value of the **kernel.shmmni** sysctl parameters on the node **ip-10-0-32-74.ec2.internal**:

```
$ oc debug node/ip-10-0-26-151.ec2.internal -q -- chroot host sysctl kernel.shmmni
```

#### Example output

```
kernel.shmmni = 4096
```

- Create a profile patch, for example, **perf-patch.yaml** that changes the **kernel.shmmni** sysctl parameter to **8192**. Defer the application of the change to a new manual restart by using the **always** method by applying the following configuration:

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
 name: performance-patch
 namespace: openshift-cluster-node-tuning-operator
 annotations:
 tuned.openshift.io/deferred: "always"
spec:
 profile:
 - name: performance-patch
 data: |
 [main]
 summary=Configuration changes profile inherited from performance created tuned
 include=openshift-node-performance-performance ①
 [sysctl]
 kernel.shmmni=8192 ②
 recommend:
 - machineConfigLabels:
 machineconfiguration.openshift.io/role: worker-cnf ③
 priority: 19
 profile: performance-patch
```

- ①** The **include** directive is used to inherit the **openshift-node-performance-performance** profile. This is a best practice to ensure that the profile is not missing any required settings.
- ②** The **kernel.shmmni** sysctl parameter is being changed to **8192**.

- 3** The **machineConfigLabels** field is used to target the **worker-cnf** role.

- Apply the profile patch by running the following command:

```
$ oc apply -f perf-patch.yaml
```

- Run the following command to verify that the profile patch is waiting for the next node restart:

```
$ oc -n openshift-cluster-node-tuning-operator get profile
```

#### Example output

NAME	TUNED	APPLIED	DEGRADED	MESSAGE
ip-10-0-26-151.ec2.internal	performance-patch	False	True	The TuneD daemon profile is waiting for the next node restart: performance-patch 126m
ip-10-0-46-60.ec2.internal	openshift-node	True	False	TuneD profile applied. 125m
ip-10-0-52-141.ec2.internal	openshift-control-plane	True	False	TuneD profile applied. 130m
ip-10-0-6-97.ec2.internal	openshift-control-plane	True	False	TuneD profile applied. 130m
ip-10-0-86-145.ec2.internal	openshift-node	True	False	TuneD profile applied. 126m
ip-10-0-92-228.ec2.internal	openshift-control-plane	True	False	TuneD profile applied. 130m

- Confirm the value of the **kernel.shmmni** sysctl parameter remain unchanged before a restart:

- Run the following command to confirm that the application of the **performance-patch** change to the **kernel.shmmni** sysctl parameter on the node **ip-10-0-26-151.ec2.internal** is not applied:

```
$ oc debug node/ip-10-0-26-151.ec2.internal -q -- chroot host sysctl kernel.shmmni
```

#### Example output

```
kernel.shmmni = 4096
```

- Restart the node **ip-10-0-26-151.ec2.internal** to apply the required changes by running the following command:

```
$ oc debug node/ip-10-0-26-151.ec2.internal -q -- chroot host reboot&
```

- In another terminal window, run the following command to verify that the node has restarted:

```
$ watch oc get nodes
```

Wait for the node **ip-10-0-26-151.ec2.internal** to transition back to the **Ready** state.

- Run the following command to verify that the profile patch is waiting for the next node restart:

```
$ oc -n openshift-cluster-node-tuning-operator get profile
```

### Example output

NAME	TUNED	APPLIED	DEGRADED	MESSAGE
AGE				
ip-10-0-20-251.ec2.internal	performance-patch applied.	True 3h3m	False	TuneD profile
ip-10-0-30-148.ec2.internal	openshift-control-plane applied.	True 3h8m	False	TuneD profile
ip-10-0-32-74.ec2.internal	openshift-node 179m	True	True	TuneD profile applied.
ip-10-0-33-49.ec2.internal	openshift-control-plane 3h8m	True	False	TuneD profile applied.
ip-10-0-84-72.ec2.internal	openshift-control-plane 3h8m	True	False	TuneD profile applied.
ip-10-0-93-89.ec2.internal	openshift-node 179m	True	False	TuneD profile applied.

12. Check that the value of the **kernel.shmmni** sysctl parameter have changed after the restart:

- Run the following command to verify that the **kernel.shmmni** sysctl parameter change has been applied on the node **ip-10-0-32-74.ec2.internal**:

```
$ oc debug node/ip-10-0-32-74.ec2.internal -q -- chroot host sysctl kernel.shmmni
```

### Example output

```
kernel.shmmni = 8192
```



### NOTE

An additional restart results in the restoration of the original value of the **kernel.shmmni** sysctl parameter.

## 10.8. 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
- eepc\_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](#)

## 10.9. CONFIGURING NODE TUNING IN A HOSTED CLUSTER

To set node-level tuning on the nodes in your hosted cluster, you can use the Node Tuning Operator. In hosted control planes, you can configure node tuning by creating config maps that contain **Tuned** objects and referencing those config maps in your node pools.

#### Procedure

1. Create a config map that contains a valid tuned manifest, and reference the manifest in a node pool. In the following example, a **Tuned** manifest defines a profile that sets **vm.dirty\_ratio** to 55 on nodes that contain the **tuned-1-node-label** node label with any value. Save the following **ConfigMap** manifest in a file named **tuned-1.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: tuned-1
 namespace: clusters
data:
```

```
tuning: |
 apiVersion: tuned.openshift.io/v1
 kind: Tuned
 metadata:
 name: tuned-1
 namespace: openshift-cluster-node-tuning-operator
 spec:
 profile:
 - data: |
 [main]
 summary=Custom OpenShift profile
 include=openshift-node
 [sysctl]
 vm.dirty_ratio="55"
 name: tuned-1-profile
 recommend:
 - priority: 20
 profile: tuned-1-profile
```



## NOTE

If you do not add any labels to an entry in the **spec.recommend** section of the Tuned spec, node-pool-based matching is assumed, so the highest priority profile in the **spec.recommend** section is applied to nodes in the pool. Although you can achieve more fine-grained node-label-based matching by setting a label value in the Tuned **.spec.recommend.match** section, node labels will not persist during an upgrade unless you set the **.spec.management.upgradeType** value of the node pool to **InPlace**.

2. Create the **ConfigMap** object in the management cluster:

```
$ oc --kubeconfig="$MGMT_KUBECONFIG" create -f tuned-1.yaml
```

3. Reference the **ConfigMap** object in the **spec.tuningConfig** field of the node pool, either by editing a node pool or creating one. In this example, assume that you have only one **NodePool**, named **nodepool-1**, which contains 2 nodes.

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: NodePool
metadata:
...
 name: nodepool-1
 namespace: clusters
...
spec:
...
 tuningConfig:
 - name: tuned-1
status:
...
```

**NOTE**

You can reference the same config map in multiple node pools. In hosted control planes, the Node Tuning Operator appends a hash of the node pool name and namespace to the name of the Tuned CRs to distinguish them. Outside of this case, do not create multiple TuneD profiles of the same name in different Tuned CRs for the same hosted cluster.

**Verification**

Now that you have created the **ConfigMap** object that contains a **Tuned** manifest and referenced it in a **NodePool**, the Node Tuning Operator syncs the **Tuned** objects into the hosted cluster. You can verify which **Tuned** objects are defined and which TuneD profiles are applied to each node.

1. List the **Tuned** objects in the hosted cluster:

```
$ oc --kubeconfig="$HC_KUBECONFIG" get tuned.tuned.openshift.io \
-n openshift-cluster-node-tuning-operator
```

**Example output**

NAME	AGE
default	7m36s
rendered	7m36s
tuned-1	65s

2. List the **Profile** objects in the hosted cluster:

```
$ oc --kubeconfig="$HC_KUBECONFIG" get profile.tuned.openshift.io \
-n openshift-cluster-node-tuning-operator
```

**Example output**

NAME	TUNED	APPLIED	DEGRADED	AGE
nodepool-1-worker-1	tuned-1-profile	True	False	7m43s
nodepool-1-worker-2	tuned-1-profile	True	False	7m14s

**NOTE**

If no custom profiles are created, the **openshift-node** profile is applied by default.

3. To confirm that the tuning was applied correctly, start a debug shell on a node and check the sysctl values:

```
$ oc --kubeconfig="$HC_KUBECONFIG" \
debug node/nodepool-1-worker-1 -- chroot /host sysctl vm.dirty_ratio
```

**Example output**

```
vm.dirty_ratio = 55
```

## 10.10. ADVANCED NODE TUNING FOR HOSTED CLUSTERS BY SETTING KERNEL BOOT PARAMETERS

For more advanced tuning in hosted control planes, which requires setting kernel boot parameters, you can also use the Node Tuning Operator. The following example shows how you can create a node pool with huge pages reserved.

### Procedure

1. Create a **ConfigMap** object that contains a **Tuned** object manifest for creating 10 huge pages that are 2 MB in size. Save this **ConfigMap** manifest in a file named **tuned-hugepages.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: tuned-hugepages
 namespace: clusters
data:
 tuning: |
 apiVersion: tuned.openshift.io/v1
 kind: Tuned
 metadata:
 name: hugepages
 namespace: openshift-cluster-node-tuning-operator
 spec:
 profile:
 - data: |
 [main]
 summary=Boot time configuration for hugepages
 include=openshift-node
 [bootloader]
 cmdline.openshift_node_hugepages=hugepagesz=2M hugepages=50
 name: openshift-node-hugepages
 recommend:
 - priority: 20
 profile: openshift-node-hugepages
```



### NOTE

The **.spec.recommend.match** field is intentionally left blank. In this case, this **Tuned** object is applied to all nodes in the node pool where this **ConfigMap** object is referenced. Group nodes with the same hardware configuration into the same node pool. Otherwise, TuneD operands can calculate conflicting kernel parameters for two or more nodes that share the same node pool.

2. Create the **ConfigMap** object in the management cluster:

```
$ oc --kubeconfig=<management_cluster_kubeconfig> create -f tuned-hugepages.yaml ①
```

- 1 Replace **<management\_cluster\_kubeconfig>** with the name of your management cluster **kubeconfig** file.

3. Create a **NodePool** manifest YAML file, customize the upgrade type of the **NodePool**, and

reference the **ConfigMap** object that you created in the **spec.tuningConfig** section. Create the **NodePool** manifest and save it in a file named **hugepages-nodepool.yaml** by using the **hcp** CLI:

```
$ hcp create nodepool aws \
--cluster-name <hosted_cluster_name> \①
--name <nodepool_name> \②
--node-count <nodepool_replicas> \③
--instance-type <instance_type> \④
--render > hugepages-nodepool.yaml
```

- ① Replace **<hosted\_cluster\_name>** with the name of your hosted cluster.
- ② Replace **<nodepool\_name>** with the name of your node pool.
- ③ Replace **<nodepool\_replicas>** with the number of your node pool replicas, for example, **2**.
- ④ Replace **<instance\_type>** with the instance type, for example, **m5.2xlarge**.



#### NOTE

The **--render** flag in the **hcp create** command does not render the secrets. To render the secrets, you must use both the **--render** and the **--render-sensitive** flags in the **hcp create** command.

4. In the **hugepages-nodepool.yaml** file, set **.spec.management.upgradeType** to **InPlace**, and set **.spec.tuningConfig** to reference the **tuned-hugepages ConfigMap** object that you created.

```
apiVersion: hypershift.openshift.io/v1alpha1
kind: NodePool
metadata:
 name: hugepages-nodepool
 namespace: clusters
...
spec:
 management:
 ...
 upgradeType: InPlace
 ...
 tuningConfig:
 - name: tuned-hugepages
```



#### NOTE

To avoid the unnecessary re-creation of nodes when you apply the new **MachineConfig** objects, set **.spec.management.upgradeType** to **InPlace**. If you use the **Replace** upgrade type, nodes are fully deleted and new nodes can replace them when you apply the new kernel boot parameters that the TuneD operand calculated.

5. Create the **NodePool** in the management cluster:

```
$ oc --kubeconfig=<management_cluster_kubeconfig> create -f hugepages-nodepool.yaml
```

## Verification

After the nodes are available, the containerized TuneD daemon calculates the required kernel boot parameters based on the applied TuneD profile. After the nodes are ready and reboot once to apply the generated **MachineConfig** object, you can verify that the TuneD profile is applied and that the kernel boot parameters are set.

1. List the **Tuned** objects in the hosted cluster:

```
$ oc --kubeconfig=<hosted_cluster_kubeconfig> get tuned.tuned.openshift.io \
-n openshift-cluster-node-tuning-operator
```

### Example output

NAME	AGE
default	123m
hugepages-8dfb1fed	1m23s
rendered	123m

2. List the **Profile** objects in the hosted cluster:

```
$ oc --kubeconfig=<hosted_cluster_kubeconfig> get profile.tuned.openshift.io \
-n openshift-cluster-node-tuning-operator
```

### Example output

NAME	TUNED	APPLIED	DEGRADED	AGE
nodepool-1-worker-1	openshift-node	True	False	132m
nodepool-1-worker-2	openshift-node	True	False	131m
hugepages-nodepool-worker-1	openshift-node-hugepages	True	False	4m8s
hugepages-nodepool-worker-2	openshift-node-hugepages	True	False	3m57s

Both of the worker nodes in the new **NodePool** have the **openshift-node-hugepages** profile applied.

3. To confirm that the tuning was applied correctly, start a debug shell on a node and check **/proc/cmdline**.

```
$ oc --kubeconfig=<hosted_cluster_kubeconfig> \
debug node/nodepool-1-worker-1 -- chroot /host cat /proc/cmdline
```

### Example output

```
BOOT_IMAGE=(hd0,gpt3)/ostree/rhcos-... hugepagesz=2M hugepages=50
```

## Additional resources

- [Hosted control planes overview](#)

# CHAPTER 11. USING CPU MANAGER AND TOPOLOGY MANAGER

CPU Manager manages groups of CPUs and constrains workloads to specific CPUs.

CPU Manager is useful for workloads that have some of these attributes:

- Require as much CPU time as possible.
- Are sensitive to processor cache misses.
- Are low-latency network applications.
- Coordinate with other processes and benefit from sharing a single processor cache.

Topology Manager collects hints from the CPU Manager, Device Manager, and other Hint Providers to align pod resources, such as CPU, SR-IOV VF, and other device resources, for all Quality of Service (QoS) classes on the same non-uniform memory access (NUMA) node.

Topology Manager uses topology information from the collected hints to decide if a pod can be accepted or rejected on a node, based on the configured Topology Manager policy and pod resources requested.

Topology Manager is useful for workloads that use hardware accelerators to support latency-critical execution and high throughput parallel computation.

To use Topology Manager you must configure CPU Manager with the **static** policy.

## 11.1. SETTING UP CPU MANAGER

To configure CPU manager, create a KubeletConfig custom resource (CR) and apply it to the desired set of nodes.

### Procedure

1. Label a node by running the following command:

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

2. To enable CPU Manager for all compute nodes, edit the CR by running the following command:

```
oc edit machineconfigpool worker
```

3. Add the **custom-kubelet: cpumanager-enabled** label to **metadata.labels** section.

```
metadata:
 creationTimestamp: 2020-xx-xx
 generation: 3
 labels:
 custom-kubelet: cpumanager-enabled
```

4. Create a **KubeletConfig**, **cpumanager-kubeletconfig.yaml**, custom resource (CR). Refer to the label created in the previous step to have the correct nodes updated with the new kubelet config. See the **machineConfigPoolSelector** section:

```

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
 name: cpumanager-enabled
spec:
 machineConfigPoolSelector:
 matchLabels:
 custom-kubelet: cpumanager-enabled
 kubeletConfig:
 cpuManagerPolicy: static ①
 cpuManagerReconcilePeriod: 5s ②

```

① Specify a policy:

- **none**. This policy explicitly enables the existing default CPU affinity scheme, providing no affinity beyond what the scheduler does automatically. This is the default policy.
- **static**. This policy allows containers in guaranteed pods with integer CPU requests. It also limits access to exclusive CPUs on the node. If **static**, you must use a lowercase **s**.

② Optional. Specify the CPU Manager reconcile frequency. The default is **5s**.

5. Create the dynamic kubelet config by running the following command:

```
oc create -f cpumanager-kubeletconfig.yaml
```

This adds the CPU Manager feature to the kubelet config and, if needed, the Machine Config Operator (MCO) reboots the node. To enable CPU Manager, a reboot is not needed.

6. Check for the merged kubelet config by running the following command:

```
oc get machineconfig 99-worker-XXXXXX-XXXXX-XXXX-XXXXX-kubelet -o json | grep ownerReference -A7
```

### Example output

```

"ownerReferences": [
 {
 "apiVersion": "machineconfiguration.openshift.io/v1",
 "kind": "KubeletConfig",
 "name": "cpumanager-enabled",
 "uid": "7ed5616d-6b72-11e9-aae1-021e1ce18878"
 }
]

```

7. Check the compute node for the updated **kubelet.conf** file by running the following command:

```
oc debug node/perf-node.example.com
sh-4.2# cat /host/etc/kubernetes/kubelet.conf | grep cpuManager
```

### Example output

```
cpuManagerPolicy: static ①
cpuManagerReconcilePeriod: 5s ②
```

- ① **cpuManagerPolicy** is defined when you create the **KubeletConfig** CR.
- ② **cpuManagerReconcilePeriod** is defined when you create the **KubeletConfig** CR.

8. Create a project by running the following command:

```
$ oc new-project <project_name>
```

9. Create a pod that requests a core or multiple cores. Both limits and requests must have their CPU value set to a whole integer. That is the number of cores that will be dedicated to this pod:

```
cat cpumanager-pod.yaml
```

### Example output

```
apiVersion: v1
kind: Pod
metadata:
 generateName: cpumanager-
spec:
 securityContext:
 runAsNonRoot: true
 seccompProfile:
 type: RuntimeDefault
 containers:
 - name: cpumanager
 image: gcr.io/google_containers/pause:3.2
 resources:
 requests:
 cpu: 1
 memory: "1G"
 limits:
 cpu: 1
 memory: "1G"
 securityContext:
 allowPrivilegeEscalation: false
 capabilities:
 drop: [ALL]
 nodeSelector:
 cpumanager: "true"
```

10. Create the pod:

```
oc create -f cpumanager-pod.yaml
```

### Verification

1. Verify that the pod is scheduled to the node that you labeled by running the following command:

```
oc describe pod cpumanager
```

### Example output

```
Name: cpumanager-6cqz7
Namespace: default
Priority: 0
PriorityClassName: <none>
Node: perf-node.example.com/xxx.xx.xx.xxx
...
Limits:
 cpu: 1
 memory: 1G
Requests:
 cpu: 1
 memory: 1G
...
QoS Class: Guaranteed
Node-Selectors: cpumanager=true
```

- Verify that a CPU has been exclusively assigned to the pod by running the following command:

```
oc describe node --selector='cpumanager=true' | grep -i cpumanager- -B2
```

### Example output

NAMESPACE	NAME	CPU Requests	CPU Limits	Memory Requests	Memory
Limits					
cpuman	cpumanager-mlrrz	1 (28%)	1 (28%)	1G (13%)	1G (13%)
					27m

- Verify that the **cgroups** are set up correctly. Get the process ID (PID) of the **pause** process by running the following commands:

```
oc debug node/perf-node.example.com
```

```
sh-4.2# systemctl status | grep -B5 pause
```



### NOTE

If the output returns multiple pause process entries, you must identify the correct pause process.

### Example output

```
└─init.scope
 └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 17
 └─kubepods.slice
 ├─kubepods-pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice
 ├─crio-b5437308f1a574c542bdf08563b865c0345c8f8c0b0a655612c.scope
 └─32706 /pause
```

- Verify that pods of quality of service (QoS) tier **Guaranteed** are placed within the **kubepods.slice** subdirectory by running the following commands:

```
cd /sys/fs/cgroup/kubepods.slice/kubepods-
pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice/crio-
b5437308f1ad1a7db0574c542bdf08563b865c0345c86e9585f8c0b0a655612c.scope
```

```
for i in `ls cpuset.cpus cgroup.procs` ; do echo -n "$i "; cat $i ; done
```



#### NOTE

Pods of other QoS tiers end up in child **cgroups** of the parent **kubepods**.

#### Example output

```
cpuset.cpus 1
tasks 32706
```

- Check the allowed CPU list for the task by running the following command:

```
grep ^Cpus_allowed_list /proc/32706/status
```

#### Example output

```
Cpus_allowed_list: 1
```

- Verify that another pod on the system cannot run on the core allocated for the **Guaranteed** pod. For example, to verify the pod in the **besteffort** QoS tier, run the following commands:

```
cat /sys/fs/cgroup/kubepods.slice/kubepods-besteffort.slice/kubepods-besteffort-
podc494a073_6b77_11e9_98c0_06bba5c387ea.slice/crio-
c56982f57b75a2420947f0afc6cafe7534c5734efc34157525fa9abbff99e3849.scope/cpus
```

```
oc describe node perf-node.example.com
```

#### Example output

```
...
Capacity:
attachable-volumes-aws-ebs: 39
cpu: 2
ephemeral-storage: 124768236Ki
hugepages-1Gi: 0
hugepages-2Mi: 0
memory: 8162900Ki
pods: 250
Allocatable:
attachable-volumes-aws-ebs: 39
cpu: 1500m
ephemeral-storage: 124768236Ki
hugepages-1Gi: 0
```

hugepages-2Mi:	0	
memory:	7548500Ki	
pods:	250	
<hr/>		
-		
default	cpumanager-6cqz7	
1G (12%)	29m	
<hr/>		
Allocated resources:		
(Total limits may be over 100 percent, i.e., overcommitted.)		
Resource	Requests	Limits
<hr/>		
cpu	1440m (96%)	1 (66%)

This VM has two CPU cores. The **system-reserved** setting reserves 500 millicores, meaning that half of one core is subtracted from the total capacity of the node to arrive at the **Node Allocatable** amount. You can see that **Allocatable CPU** is 1500 millicores. This means you can run one of the CPU Manager pods since each will take one whole core. A whole core is equivalent to 1000 millicores. If you try to schedule a second pod, the system will accept the pod, but it will never be scheduled:

NAME	READY	STATUS	RESTARTS	AGE
cpumanager-6cqz7	1/1	Running	0	33m
cpumanager-7qc2t	0/1	Pending	0	11s

## 11.2. TOPOLOGY MANAGER POLICIES

Topology Manager aligns **Pod** resources of all Quality of Service (QoS) classes by collecting topology hints from Hint Providers, such as CPU Manager and Device Manager, and using the collected hints to align the **Pod** resources.

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

### none policy

This is the default policy and does not perform any topology alignment.

### best-effort policy

For each container in a pod with the **best-effort** topology management policy, kubelet calls each Hint Provider to discover their resource availability. Using this information, the Topology Manager stores the preferred NUMA Node affinity for that container. If the affinity is not preferred, Topology Manager stores this and admits the pod to the node.

### restricted policy

For each container in a pod with the **restricted** topology management policy, kubelet calls each Hint Provider to discover their resource availability. Using this information, the Topology Manager stores the preferred NUMA Node affinity for that container. If the affinity is not preferred, Topology Manager rejects this pod from the node, resulting in a pod in a **Terminated** state with a pod admission failure.

### single-numa-node policy

For each container in a pod with the **single-numa-node** topology management policy, kubelet calls each Hint Provider to discover their resource availability. Using this information, the Topology Manager determines if a single NUMA Node affinity is possible. If it is, the pod is admitted to the

node. If a single NUMA Node affinity is not possible, the Topology Manager rejects the pod from the node. This results in a pod in a Terminated state with a pod admission failure.

## 11.3. SETTING UP TOPOLOGY MANAGER

To use Topology Manager, you must configure an allocation policy in the **KubeletConfig** custom resource (CR) named **cpumanager-enabled**. This file might exist if you have set up CPU Manager. If the file does not exist, you can create the file.

### Prerequisites

- Configure the CPU Manager policy to be **static**.

### Procedure

To activate Topology Manager:

- Configure the Topology Manager allocation policy in the custom resource.

```
$ oc edit KubeletConfig cpumanager-enabled

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
 name: cpumanager-enabled
spec:
 machineConfigPoolSelector:
 matchLabels:
 custom-kubelet: cpumanager-enabled
 kubeletConfig:
 cpuManagerPolicy: static 1
 cpuManagerReconcilePeriod: 5s
 topologyManagerPolicy: single-numa-node 2
```

- 1** This parameter must be **static** with a lowercase **s**.
- 2** Specify your selected Topology Manager allocation policy. Here, the policy is **single-numa-node**. Acceptable values are: **default**, **best-effort**, **restricted**, **single-numa-node**.

## 11.4. POD INTERACTIONS WITH TOPOLOGY MANAGER POLICIES

The example **Pod** specs below help illustrate pod interactions with Topology Manager.

The following pod runs in the **BestEffort** QoS class because no resource requests or limits are specified.

```
spec:
 containers:
 - name: nginx
 image: nginx
```

The next pod runs in the **Burstable** QoS class because requests are less than limits.

```
spec:
```

```
containers:
- name: nginx
 image: nginx
 resources:
 limits:
 memory: "200Mi"
 requests:
 memory: "100Mi"
```

If the selected policy is anything other than **none**, Topology Manager would not consider either of these **Pod** specifications.

The last example pod below runs in the Guaranteed QoS class because requests are equal to limits.

```
spec:
 containers:
 - name: nginx
 image: nginx
 resources:
 limits:
 memory: "200Mi"
 cpu: "2"
 example.com/device: "1"
 requests:
 memory: "200Mi"
 cpu: "2"
 example.com/device: "1"
```

Topology Manager would consider this pod. The Topology Manager would consult the hint providers, which are CPU Manager and Device Manager, to get topology hints for the pod.

Topology Manager will use this information to store the best topology for this container. In the case of this pod, CPU Manager and Device Manager will use this stored information at the resource allocation stage.

# CHAPTER 12. SCHEDULING NUMA-AWARE WORKLOADS

Learn about NUMA-aware scheduling and how you can use it to deploy high performance workloads in an OpenShift Container Platform cluster.

The NUMA Resources Operator allows you to schedule high-performance workloads in the same NUMA zone. It deploys a node resources exporting agent that reports on available cluster node NUMA resources, and a secondary scheduler that manages the workloads.

## 12.1. ABOUT NUMA-AWARE SCHEDULING

### Introduction to NUMA

Non-Uniform Memory Access (NUMA) is a compute platform architecture that allows different CPUs to access different regions of memory at different speeds. NUMA resource topology refers to the locations of CPUs, memory, and PCI devices relative to each other in the compute node. Colocated resources are said to be in the same *NUMA zone*. For high-performance applications, the cluster needs to process pod workloads in a single NUMA zone.

### Performance considerations

NUMA architecture allows a CPU with multiple memory controllers to use any available memory across CPU complexes, regardless of where the memory is located. This allows for increased flexibility at the expense of performance. A CPU processing a workload using memory that is outside its NUMA zone is slower than a workload processed in a single NUMA zone. Also, for I/O-constrained workloads, the network interface on a distant NUMA zone slows down how quickly information can reach the application. High-performance workloads, such as telecommunications workloads, cannot operate to specification under these conditions.

### NUMA-aware scheduling

NUMA-aware scheduling aligns the requested cluster compute resources (CPUs, memory, devices) in the same NUMA zone to process latency-sensitive or high-performance workloads efficiently. NUMA-aware scheduling also improves pod density per compute node for greater resource efficiency.

### Integration with Node Tuning Operator

By integrating the Node Tuning Operator’s performance profile with NUMA-aware scheduling, you can further configure CPU affinity to optimize performance for latency-sensitive workloads.

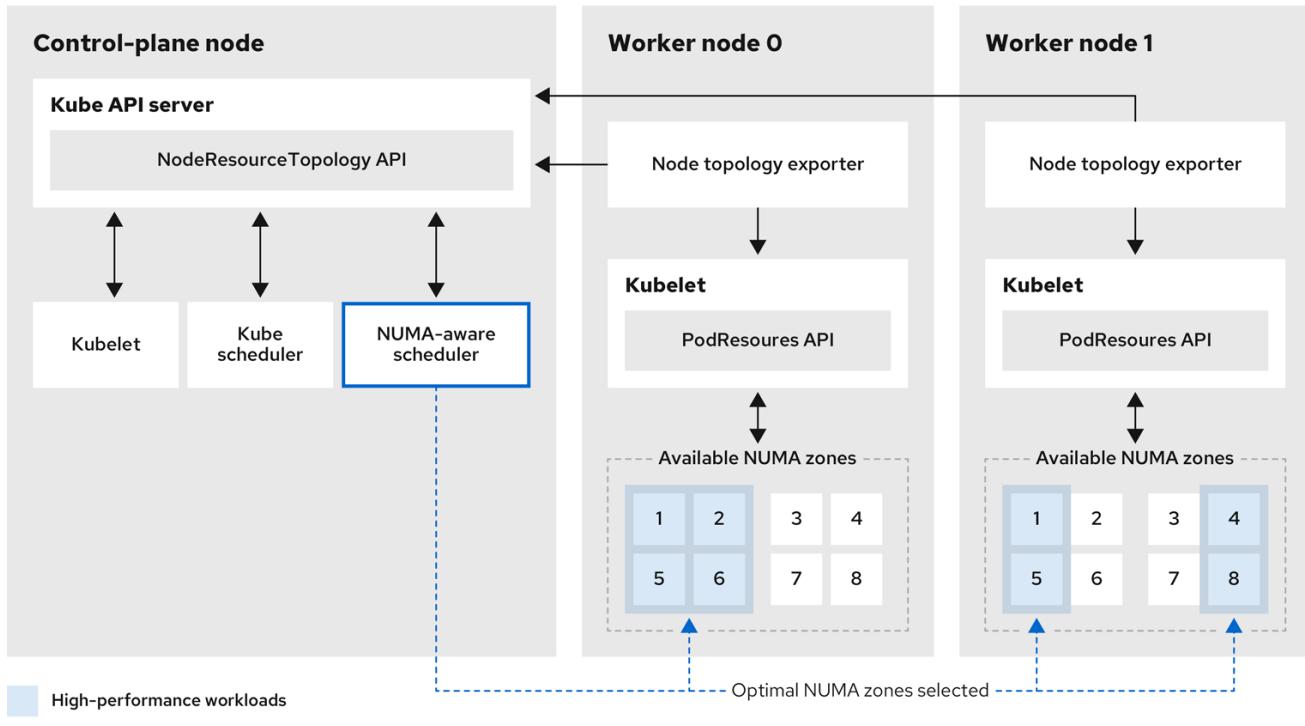
### Default scheduling logic

The default OpenShift Container Platform pod scheduler scheduling logic considers the available resources of the entire compute node, not individual NUMA zones. If the most restrictive resource alignment is requested in the kubelet topology manager, error conditions can occur when admitting the pod to a node. Conversely, if the most restrictive resource alignment is not requested, the pod can be admitted to the node without proper resource alignment, leading to worse or unpredictable performance. For example, runaway pod creation with **Topology Affinity Error** statuses can occur when the pod scheduler makes suboptimal scheduling decisions for guaranteed pod workloads without knowing if the pod’s requested resources are available. Scheduling mismatch decisions can cause indefinite pod startup delays. Also, depending on the cluster state and resource allocation, poor pod scheduling decisions can cause extra load on the cluster because of failed startup attempts.

### NUMA-aware pod scheduling diagram

The NUMA Resources Operator deploys a custom NUMA resources secondary scheduler and other resources to mitigate against the shortcomings of the default OpenShift Container Platform pod scheduler. The following diagram provides a high-level overview of NUMA-aware pod scheduling.

Figure 12.1. NUMA-aware scheduling overview



216\_OpenShift\_0222

### NodeResourceTopology API

The **NodeResourceTopology** API describes the available NUMA zone resources in each compute node.

### NUMA-aware scheduler

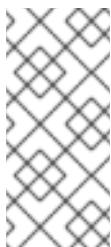
The NUMA-aware secondary scheduler receives information about the available NUMA zones from the **NodeResourceTopology** API and schedules high-performance workloads on a node where it can be optimally processed.

### Node topology exporter

The node topology exporter exposes the available NUMA zone resources for each compute node to the **NodeResourceTopology** API. The node topology exporter daemon tracks the resource allocation from the kubelet by using the **PodResources** API.

### PodResources API

The **PodResources** API is local to each node and exposes the resource topology and available resources to the kubelet.



### NOTE

The **List** endpoint of the **PodResources** API exposes exclusive CPUs allocated to a particular container. The API does not expose CPUs that belong to a shared pool.

The **GetAllocatableResources** endpoint exposes allocatable resources available on a node.

## 12.2. NUMA RESOURCE SCHEDULING STRATEGIES

When scheduling high-performance workloads, the secondary scheduler can employ different strategies

to determine which NUMA node within a chosen worker node will handle the workload. The supported strategies in OpenShift Container Platform include **LeastAllocated**, **MostAllocated**, and **BalancedAllocation**. Understanding these strategies helps optimize workload placement for performance and resource utilization.

When a high-performance workload is scheduled in a NUMA-aware cluster, the following steps occur:

1. The scheduler first selects a suitable worker node based on cluster-wide criteria. For example taints, labels, or resource availability.
2. After a worker node is selected, the scheduler evaluates its NUMA nodes and applies a scoring strategy to decide which NUMA node will handle the workload.
3. After a workload is scheduled, the selected NUMA node's resources are updated to reflect the allocation.

The default strategy applied is the **LeastAllocated** strategy. This assigns workloads to the NUMA node with the most available resources that is the least utilized NUMA node. The goal of this strategy is to spread workloads across NUMA nodes to reduce contention and avoid hotspots.

The following table summarizes the different strategies and their outcomes:

### Scoring strategy summary

**Table 12.1. Scoring strategy summary**

Strategy	Description	Outcome
<b>LeastAllocated</b>	Favors NUMA nodes with the most available resources.	Spreads workloads to reduce contention and ensure headroom for high-priority tasks.
<b>MostAllocated</b>	Favors NUMA nodes with the least available resources.	Consolidates workloads on fewer NUMA nodes, freeing others for energy efficiency.
<b>BalancedAllocation</b>	Favors NUMA nodes with balanced CPU and memory usage.	Ensures even resource utilization, preventing skewed usage patterns.

### LeastAllocated strategy example

The **LeastAllocated** is the default strategy. This strategy assigns workloads to the NUMA node with the most available resources, minimizing resource contention and spreading workloads across NUMA nodes. This reduces hotspots and ensures sufficient headroom for high-priority tasks. Assume a worker node has two NUMA nodes, and the workload requires 4 vCPUs and 8 GB of memory:

**Table 12.2. Example initial NUMA nodes state**

NUMA node	Total CPUs	Used CPUs	Total memory (GB)	Used memory (GB)	Available resources

NUMA node	Total CPUs	Used CPUs	Total memory (GB)	Used memory (GB)	Available resources
NUMA 1	16	12	64	56	4 CPUs, 8 GB memory
NUMA 2	16	6	64	24	10 CPUs, 40 GB memory

Because NUMA 2 has more available resources compared to NUMA 1, the workload is assigned to NUMA 2.

### MostAllocated strategy example

The **MostAllocated** strategy consolidates workloads by assigning them to the NUMA node with the least available resources, which is the most utilized NUMA node. This approach helps free other NUMA nodes for energy efficiency or critical workloads requiring full isolation. This example uses the "Example initial NUMA nodes state" values listed in the **LeastAllocated** section.

The workload again requires 4 vCPUs and 8 GB memory. NUMA 1 has fewer available resources compared to NUMA 2, so the scheduler assigns the workload to NUMA 1, further utilizing its resources while leaving NUMA 2 idle or minimally loaded.

### BalancedAllocation strategy example

The **BalancedAllocation** strategy assigns workloads to the NUMA node with the most balanced resource utilization across CPU and memory. The goal is to prevent imbalanced usage, such as high CPU utilization with underutilized memory. Assume a worker node has the following NUMA node states:

Table 12.3. Example NUMA nodes initial state for **BalancedAllocation**

NUMA node	CPU usage	Memory usage	BalancedAllocation score
NUMA 1	60%	55%	High (more balanced)
NUMA 2	80%	20%	Low (less balanced)

NUMA 1 has a more balanced CPU and memory utilization compared to NUMA 2 and therefore, with the **BalancedAllocation** strategy in place, the workload is assigned to NUMA 1.

### Additional resources

- [Scheduling pods using a secondary scheduler](#)
- [Changing where high-performance workloads run](#)

## 12.3. INSTALLING THE NUMA RESOURCES OPERATOR

NUMA Resources Operator deploys resources that allow you to schedule NUMA-aware workloads and deployments. You can install the NUMA Resources Operator using the OpenShift Container Platform CLI or the web console.

### 12.3.1. Installing the NUMA Resources Operator using the CLI

As a cluster administrator, you can install the Operator using the CLI.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

1. Create a namespace for the NUMA Resources Operator:
  - a. Save the following YAML in the **nro-namespace.yaml** file:

```
apiVersion: v1
kind: Namespace
metadata:
 name: openshift-numaresources
```

- b. Create the **Namespace** CR by running the following command:

```
$ oc create -f nro-namespace.yaml
```

2. Create the Operator group for the NUMA Resources Operator:

- a. Save the following YAML in the **nro-operatorgroup.yaml** file:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
 name: numaresources-operator
 namespace: openshift-numaresources
spec:
 targetNamespaces:
 - openshift-numaresources
```

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

```
$ oc create -f nro-operatorgroup.yaml
```

3. Create the subscription for the NUMA Resources Operator:

- a. Save the following YAML in the **nro-sub.yaml** file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
 name: numaresources-operator
```

```

 namespace: openshift-numaresources
 spec:
 channel: "4.18"
 name: numaresources-operator
 source: redhat-operators
 sourceNamespace: openshift-marketplace

```

- Create the **Subscription** CR by running the following command:

```
$ oc create -f nro-sub.yaml
```

## Verification

- Verify that the installation succeeded by inspecting the CSV resource in the **openshift-numaresources** namespace. Run the following command:

```
$ oc get csv -n openshift-numaresources
```

### Example output

NAME	DISPLAY	VERSION	REPLACES	PHASE
numaresources-operator.v4.18.2	numaresources-operator	4.18.2		Succeeded

### 12.3.2. Installing the NUMA Resources Operator using the web console

As a cluster administrator, you can install the NUMA Resources Operator using the web console.

#### Procedure

- Create a namespace for the NUMA Resources Operator:
  - In the OpenShift Container Platform web console, click **Administration** → **Namespaces**.
  - Click **Create Namespace**, enter **openshift-numaresources** in the **Name** field, and then click **Create**.
- Install the NUMA Resources Operator:
  - In the OpenShift Container Platform web console, click **Operators** → **OperatorHub**.
  - Choose **numaresources-operator** from the list of available Operators, and then click **Install**.
  - In the **Installed Namespaces** field, select the **openshift-numaresources** namespace, and then click **Install**.
- Optional: Verify that the NUMA Resources Operator installed successfully:
  - Switch to the **Operators** → **Installed Operators** page.
  - Ensure that **NUMA Resources Operator** is listed in the **openshift-numaresources** namespace with a **Status** of **InstallSucceeded**.

**NOTE**

During installation an Operator might display a **Failed** status. If the installation later succeeds with an **InstallSucceeded** message, you can ignore the **Failed** message.

If the Operator does not appear as installed, to troubleshoot further:

- Go to the **Operators → Installed Operators** page and inspect the **Operator Subscriptions** and **Install Plans** tabs for any failure or errors under **Status**.
- Go to the **Workloads → Pods** page and check the logs for pods in the **default** project.

## 12.4. SCHEDULING NUMA-AWARE WORKLOADS

Clusters running latency-sensitive workloads typically feature performance profiles that help to minimize workload latency and optimize performance. The NUMA-aware scheduler deploys workloads based on available node NUMA resources and with respect to any performance profile settings applied to the node. The combination of NUMA-aware deployments, and the performance profile of the workload, ensures that workloads are scheduled in a way that maximizes performance.

For the NUMA Resources Operator to be fully operational, you must deploy the **NUMAResourcesOperator** custom resource and the NUMA-aware secondary pod scheduler.

### 12.4.1. Creating the **NUMAResourcesOperator** custom resource

When you have installed the NUMA Resources Operator, then create the **NUMAResourcesOperator** custom resource (CR) that instructs the NUMA Resources Operator to install all the cluster infrastructure needed to support the NUMA-aware scheduler, including daemon sets and APIs.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the NUMA Resources Operator.

#### Procedure

1. Create the **NUMAResourcesOperator** custom resource:
  - a. Save the following minimal required YAML file example as **nrop.yaml**:

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesOperator
metadata:
 name: numaresourcesoperator
spec:
 nodeGroups:
 - machineConfigPoolSelector:
 matchLabels:
 pools.operator.machineconfiguration.openshift.io/worker: ""
```

**1**

- 1** This must match the **MachineConfigPool** resource that you want to configure the NUMA Resources Operator on. For example, you might have created a

- b. Create the **NUMAResourcesOperator** CR by running the following command:

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

2. Optional: To enable NUMA-aware scheduling for multiple machine config pools (MCPs), define a separate **NodeGroup** for each pool. For example, define three **NodeGroups** for **worker-cnf**, **worker-ht**, and **worker-other**, in the **NUMAResourcesOperator** CR as shown in the following example:

#### Example YAML definition for a **NUMAResourcesOperator** CR with multiple **NodeGroups**

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesOperator
metadata:
 name: numaresourcesoperator
spec:
 logLevel: Normal
 nodeGroups:
 - machineConfigPoolSelector:
 matchLabels:
 machineconfiguration.openshift.io/role: worker-ht
 - machineConfigPoolSelector:
 matchLabels:
 machineconfiguration.openshift.io/role: worker-cnf
 - machineConfigPoolSelector:
 matchLabels:
 machineconfiguration.openshift.io/role: worker-other
```

#### Verification

1. Verify that the NUMA Resources Operator deployed successfully by running the following command:

```
$ oc get numaresourcesoperators.nodetopology.openshift.io
```

#### Example output

NAME	AGE
numaresourcesoperator	27s

2. After a few minutes, run the following command to verify that the required resources deployed successfully:

```
$ oc get all -n openshift-numaresources
```

#### Example output

NAME	READY	STATUS	RESTARTS	AGE
------	-------	--------	----------	-----

pod/numaresources-controller-manager-7d9d84c58d-qk2mr	1/1	Running	0	12m
pod/numaresourcesoperator-worker-7d96r	2/2	Running	0	97s
pod/numaresourcesoperator-worker-crsht	2/2	Running	0	97s
pod/numaresourcesoperator-worker-jp9mw	2/2	Running	0	97s

## 12.4.2. Deploying the NUMA-aware secondary pod scheduler

After you install the NUMA Resources Operator, follow this procedure to deploy the NUMA-aware secondary pod scheduler.

### Procedure

1. Create the **NUMAResourcesScheduler** custom resource that deploys the NUMA-aware custom pod scheduler:

- a. Save the following minimal required YAML in the **nro-scheduler.yaml** file:

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesScheduler
metadata:
 name: numaresourcesscheduler
spec:
 imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-rhel9:v4.18"
1
```

- 1 In a disconnected environment, make sure to configure the resolution of this image by either:
  - Creating an **ImageTagMirrorSet** custom resource (CR). For more information, see "Configuring image registry repository mirroring" in the "Additional resources" section.
  - Setting the URL to the disconnected registry.

- b. Create the **NUMAResourcesScheduler** CR by running the following command:

```
$ oc create -f nro-scheduler.yaml
```

2. After a few seconds, run the following command to confirm the successful deployment of the required resources:

```
$ oc get all -n openshift-numaresources
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE
pod/numaresources-controller-manager-7d9d84c58d-qk2mr	1/1	Running	0	12m
pod/numaresourcesoperator-worker-7d96r	2/2	Running	0	97s
pod/numaresourcesoperator-worker-crsht	2/2	Running	0	97s
pod/numaresourcesoperator-worker-jp9mw	2/2	Running	0	97s
pod/secondary-scheduler-847cb74f84-9whlm	1/1	Running	0	10m

NAME	DESIRED	CURRENT	READY	UP-TO-DATE

AVAILABLE NODE SELECTOR	AGE	3	3	3	3	node-
daemonset.apps/numaresourcesoperator-worker	3					
role.kubernetes.io/worker= 98s						
NAME	READY	UP-TO-DATE	AVAILABLE	AGE		
deployment.apps/numaresources-controller-manager	1/1	1	1	12m		
deployment.apps/secondary-scheduler	1/1	1	1	10m		
NAME	DESIRED	CURRENT	READY	AGE		
replicaset.apps/numaresources-controller-manager-7d9d84c58d	1	1	1	12m		
replicaset.apps/secondary-scheduler-847cb74f84	1	1	1	10m		

### 12.4.3. Configuring a single NUMA node policy

The NUMA Resources Operator requires a single NUMA node policy to be configured on the cluster. This can be achieved in two ways: by creating and applying a performance profile, or by configuring a KubeletConfig.



#### NOTE

The preferred way to configure a single NUMA node policy is to apply a performance profile. You can use the Performance Profile Creator (PPC) tool to create the performance profile. If a performance profile is created on the cluster, it automatically creates other tuning components like **KubeletConfig** and the **tuned** profile.

For more information about creating a performance profile, see "About the Performance Profile Creator" in the "Additional resources" section.

#### Additional resources

- [Configuring image registry repository mirroring](#)
- [About the Performance Profile Creator](#)

### 12.4.4. Sample performance profile

This example YAML shows a performance profile created by using the performance profile creator (PPC) tool:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: performance
spec:
 cpu:
 isolated: "3"
 reserved: 0-2
 machineConfigPoolSelector:
 pools.operator.machineconfiguration.openshift.io/worker: "" 1
 nodeSelector:
 node-role.kubernetes.io/worker: ""
 numa:
 topologyPolicy: single-numa-node 2
 realTimeKernel:
```

```

enabled: true
workloadHints:
 highPowerConsumption: true
 perPodPowerManagement: false
 realTime: true

```

- 1 This should match the **MachineConfigPool** that you want to configure the NUMA Resources Operator on. For example, you might have created a **MachineConfigPool** named **worker-cnf** that designates a set of nodes that run telecommunications workloads.
- 2 The **topologyPolicy** must be set to **single-numa-node**. Ensure that this is the case by setting the **topology-manager-policy** argument to **single-numa-node** when running the PPC tool.

## 12.4.5. Creating a KubeletConfig CR

The recommended way to configure a single NUMA node policy is to apply a performance profile. Another way is by creating and applying a **KubeletConfig** custom resource (CR), as shown in the following procedure.

### Procedure

- 1 Create the **KubeletConfig** custom resource (CR) that configures the pod admittance policy for the machine profile:
  - a. Save the following YAML in the **nro-kubeletconfig.yaml** file:

```

apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
 name: worker-tuning
spec:
 machineConfigPoolSelector:
 matchLabels:
 pools.operator.machineconfiguration.openshift.io/worker: "" ①
 kubeletConfig:
 cpuManagerPolicy: "static" ②
 cpuManagerReconcilePeriod: "5s"
 reservedSystemCPUs: "0,1" ③
 memoryManagerPolicy: "Static" ④
 evictionHard:
 memory.available: "100Mi"
 kubeReserved:
 memory: "512Mi"
 reservedMemory:
 - numaNode: 0
 limits:
 memory: "1124Mi"
 systemReserved:
 memory: "512Mi"
 topologyManagerPolicy: "single-numa-node" ⑤

```

- 1 Adjust this label to match the **machineConfigPoolSelector** in the **NUMAResourcesOperator** CR.

- 2 For **cpuManagerPolicy**, **static** must use a lowercase **s**.
- 3 Adjust this based on the CPU on your nodes.
- 4 For **memoryManagerPolicy**, **Static** must use an uppercase **S**.
- 5 **topologyManagerPolicy** must be set to **single-numa-node**.

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

```
$ oc create -f nro-kubeletconfig.yaml
```



#### NOTE

Applying performance profile or **KubeletConfig** automatically triggers rebooting of the nodes. If no reboot is triggered, you can troubleshoot the issue by looking at the labels in **KubeletConfig** that address the node group.

### 12.4.6. Scheduling workloads with the NUMA-aware scheduler

Now that **topo-aware-scheduler** is installed, the **NUMAResourcesOperator** and **NUMAResourcesScheduler** CRs are applied and your cluster has a matching performance profile or **kubeletconfig**, you can schedule workloads with the NUMA-aware scheduler using deployment CRs that specify the minimum required resources to process the workload.

The following example deployment uses NUMA-aware scheduling for a sample workload.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

1. Get the name of the NUMA-aware scheduler that is deployed in the cluster by running the following command:

```
$ oc get numaresourcesschedulers.nodetopology.openshift.io numaresourcesscheduler -o json | jq '.status.schedulerName'
```

#### Example output

```
"topo-aware-scheduler"
```

2. Create a **Deployment** CR that uses scheduler named **topo-aware-scheduler**, for example:

- a. Save the following YAML in the **nro-deployment.yaml** file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: numa-deployment-1
```

```

namespace: openshift-numaresources
spec:
 replicas: 1
 selector:
 matchLabels:
 app: test
 template:
 metadata:
 labels:
 app: test
 spec:
 schedulerName: topo-aware-scheduler ①
 containers:
 - name: ctnr
 image: quay.io/openshifttest/hello-openshift:openshift
 imagePullPolicy: IfNotPresent
 resources:
 limits:
 memory: "100Mi"
 cpu: "10"
 requests:
 memory: "100Mi"
 cpu: "10"
 - name: ctnr2
 image: registry.access.redhat.com/rhel:latest
 imagePullPolicy: IfNotPresent
 command: ["/bin/sh", "-c"]
 args: ["while true; do sleep 1h; done;"]
 resources:
 limits:
 memory: "100Mi"
 cpu: "8"
 requests:
 memory: "100Mi"
 cpu: "8"

```

- ① **schedulerName** must match the name of the NUMA-aware scheduler that is deployed in your cluster, for example **topo-aware-scheduler**.

- b. Create the **Deployment** CR by running the following command:

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

## Verification

- Verify that the deployment was successful:

```
$ oc get pods -n openshift-numaresources
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE
numa-deployment-1-6c4f5bdb84-wgn6g	2/2	Running	0	5m2s
numaresources-controller-manager-7d9d84c58d-4v65j	1/1	Running	0	18m

numaresourcesoperator-worker-7d96r	2/2	Running	4	43m
numaresourcesoperator-worker-crsht	2/2	Running	2	43m
numaresourcesoperator-worker-jp9mw	2/2	Running	2	43m
secondary-scheduler-847cb74f84-fpncj	1/1	Running	0	18m

2. Verify that the **topo-aware-scheduler** is scheduling the deployed pod by running the following command:

```
$ oc describe pod numa-deployment-1-6c4f5bdb84-wgn6g -n openshift-numaresources
```

#### Example output

Events:

Type	Reason	Age	From	Message
Normal	Scheduled	4m45s	topo-aware-scheduler	Successfully assigned openshift-numaresources/numa-deployment-1-6c4f5bdb84-wgn6g to worker-1



#### NOTE

Deployments that request more resources than is available for scheduling will fail with a **MinimumReplicasUnavailable** error. The deployment succeeds when the required resources become available. Pods remain in the **Pending** state until the required resources are available.

3. Verify that the expected allocated resources are listed for the node.

- a. Identify the node that is running the deployment pod by running the following command:

```
$ oc get pods -n openshift-numaresources -o wide
```

#### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
NOMINATED NODE	READYNESS GATES					
numa-deployment-1-6c4f5bdb84-wgn6g	0/2	Running	0	82m	10.128.2.50	worker-1 <none> <none>

- b. Run the following command with the name of that node that is running the deployment pod.

```
$ oc describe noderesourcetopologies.topology.node.k8s.io worker-1
```

#### Example output

...

Zones:

Costs:

Name: node-0

Value: 10

Name: node-1

Value: 21

```

Name: node-0
Resources:
 Allocatable: 39
 Available: 21 1
 Capacity: 40
 Name: cpu
 Allocatable: 6442450944
 Available: 6442450944
 Capacity: 6442450944
 Name: hugepages-1Gi
 Allocatable: 134217728
 Available: 134217728
 Capacity: 134217728
 Name: hugepages-2Mi
 Allocatable: 262415904768
 Available: 262206189568
 Capacity: 270146007040
 Name: memory
Type: Node

```

- 1 The **Available** capacity is reduced because of the resources that have been allocated to the guaranteed pod.

Resources consumed by guaranteed pods are subtracted from the available node resources listed under **noderesourcetopologies.topology.node.k8s.io**.

- Resource allocations for pods with a **Best-effort** or **Burstable** quality of service (**qosClass**) are not reflected in the NUMA node resources under **noderesourcetopologies.topology.node.k8s.io**. If a pod's consumed resources are not reflected in the node resource calculation, verify that the pod has **qosClass** of **Guaranteed** and the CPU request is an integer value, not a decimal value. You can verify the that the pod has a **qosClass** of **Guaranteed** by running the following command:

```
$ oc get pod numa-deployment-1-6c4f5bdb84-wgn6g -n openshift-numaresources -o jsonpath="{ .status.qosClass }"
```

### Example output

Guaranteed

## 12.5. OPTIONAL: CONFIGURING POLLING OPERATIONS FOR NUMA RESOURCES UPDATES

The daemons controlled by the NUMA Resources Operator in their **nodeGroup** poll resources to retrieve updates about available NUMA resources. You can fine-tune polling operations for these daemons by configuring the **spec.nodeGroups** specification in the **NUMAResourcesOperator** custom resource (CR). This provides advanced control of polling operations. Configure these specifications to improve scheduling behavior and troubleshoot suboptimal scheduling decisions.

The configuration options are the following:

- **infoRefreshMode**: Determines the trigger condition for polling the kubelet. The NUMA Resources Operator reports the resulting information to the API server.

- **infoRefreshPeriod**: Determines the duration between polling updates.
- **podsFingerprinting**: Determines if point-in-time information for the current set of pods running on a node is exposed in polling updates.

**NOTE**

The default value for **podsFingerprinting** is **EnabledExclusiveResources**. To optimize scheduler performance, set **podsFingerprinting** to either **EnabledExclusiveResources** or **Enabled**. Additionally, configure the **cacheResyncPeriod** in the **NUMAResourcesScheduler** custom resource (CR) to a value greater than 0. The **cacheResyncPeriod** specification helps to report more exact resource availability by monitoring pending resources on nodes.

**Prerequisites**

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the NUMA Resources Operator.

**Procedure**

- Configure the **spec.nodeGroups** specification in your **NUMAResourcesOperator** CR:

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesOperator
metadata:
 name: numaresourcesoperator
spec:
 nodeGroups:
 - config:
 infoRefreshMode: Periodic ①
 infoRefreshPeriod: 10s ②
 podsFingerprinting: Enabled ③
 name: worker
```

- ① Valid values are **Periodic**, **Events**, **PeriodicAndEvents**. Use **Periodic** to poll the kubelet at intervals that you define in **infoRefreshPeriod**. Use **Events** to poll the kubelet at every pod lifecycle event. Use **PeriodicAndEvents** to enable both methods.
- ② Define the polling interval for **Periodic** or **PeriodicAndEvents** refresh modes. The field is ignored if the refresh mode is **Events**.
- ③ Valid values are **Enabled**, **Disabled**, and **EnabledExclusiveResources**. Setting to **Enabled** or **EnabledExclusiveResources** is a requirement for the **cacheResyncPeriod** specification in the **NUMAResourcesScheduler**.

**Verification**

1. After you deploy the NUMA Resources Operator, verify that the node group configurations were applied by running the following command:

```
$ oc get numaresop numaresourcesoperator -o json | jq '.status'
```

### Example output

```
...
"config": {
 "infoRefreshMode": "Periodic",
 "infoRefreshPeriod": "10s",
 "podsFingerprinting": "Enabled"
},
"name": "worker"
...
```

## 12.6. TROUBLESHOOTING NUMA-AWARE SCHEDULING

To troubleshoot common problems with NUMA-aware pod scheduling, perform the following steps.

### Prerequisites

- Install the OpenShift Container Platform CLI (**oc**).
- Log in as a user with cluster-admin privileges.
- Install the NUMA Resources Operator and deploy the NUMA-aware secondary scheduler.

### Procedure

1. Verify that the **noderesourcetopologies** CRD is deployed in the cluster by running the following command:

```
$ oc get crd | grep noderesourcetopologies
```

### Example output

NAME	CREATED AT
noderesourcetopologies.topology.node.k8s.io	2022-01-18T08:28:06Z

2. Check that the NUMA-aware scheduler name matches the name specified in your NUMA-aware workloads by running the following command:

```
$ oc get numaresourcesschedulers.nodetopology.openshift.io numaresourcesscheduler -o json | jq '.status.schedulerName'
```

### Example output

```
topo-aware-scheduler
```

3. Verify that NUMA-aware schedulable nodes have the **noderesourcetopologies** CR applied to them. Run the following command:

```
$ oc get noderesourcetopologies.topology.node.k8s.io
```

### Example output

NAME	AGE
compute-0.example.com	17h
compute-1.example.com	17h



#### NOTE

The number of nodes should equal the number of worker nodes that are configured by the machine config pool (**mcp**) worker definition.

- Verify the NUMA zone granularity for all schedulable nodes by running the following command:

```
$ oc get noderesourcetopologies.topology.node.k8s.io -o yaml
```

### Example output

```
apiVersion: v1
items:
- apiVersion: topology.node.k8s.io/v1
 kind: NodeResourceTopology
 metadata:
 annotations:
 k8stopoawareschedwg/rte-update: periodic
 creationTimestamp: "2022-06-16T08:55:38Z"
 generation: 63760
 name: worker-0
 resourceVersion: "8450223"
 uid: 8b77be46-08c0-4074-927b-d49361471590
 topologyPolicies:
 - SingleNUMANodeContainerLevel
 zones:
 - costs:
 - name: node-0
 value: 10
 - name: node-1
 value: 21
 name: node-0
 resources:
 - allocatable: "38"
 available: "38"
 capacity: "40"
 name: cpu
 - allocatable: "134217728"
 available: "134217728"
 capacity: "134217728"
 name: hugepages-2Mi
 - allocatable: "262352048128"
 available: "262352048128"
 capacity: "270107316224"
 name: memory
```

```
- allocatable: "6442450944"
 available: "6442450944"
 capacity: "6442450944"
 name: hugepages-1Gi
type: Node
- costs:
 - name: node-0
 value: 21
 - name: node-1
 value: 10
name: node-1
resources:
- allocatable: "268435456"
 available: "268435456"
 capacity: "268435456"
 name: hugepages-2Mi
- allocatable: "269231067136"
 available: "269231067136"
 capacity: "270573244416"
 name: memory
- allocatable: "40"
 available: "40"
 capacity: "40"
 name: cpu
- allocatable: "1073741824"
 available: "1073741824"
 capacity: "1073741824"
 name: hugepages-1Gi
type: Node
- apiVersion: topology.node.k8s.io/v1
kind: NodeResourceTopology
metadata:
 annotations:
 k8stopoawareschedwg/rte-update: periodic
 creationTimestamp: "2022-06-16T08:55:37Z"
 generation: 62061
 name: worker-1
 resourceVersion: "8450129"
 uid: e8659390-6f8d-4e67-9a51-1ea34bba1cc3
 topologyPolicies:
 - SingleNUMANodeContainerLevel
 zones: ①
 - costs:
 - name: node-0
 value: 10
 - name: node-1
 value: 21
 name: node-0
 resources: ②
 - allocatable: "38"
 available: "38"
 capacity: "40"
 name: cpu
 - allocatable: "6442450944"
 available: "6442450944"
 capacity: "6442450944"
```

```

name: hugepages-1Gi
- allocatable: "134217728"
 available: "134217728"
 capacity: "134217728"
name: hugepages-2Mi
- allocatable: "262391033856"
 available: "262391033856"
 capacity: "270146301952"
name: memory
type: Node
- costs:
 - name: node-0
 value: 21
 - name: node-1
 value: 10
name: node-1
resources:
- allocatable: "40"
 available: "40"
 capacity: "40"
name: cpu
- allocatable: "1073741824"
 available: "1073741824"
 capacity: "1073741824"
name: hugepages-1Gi
- allocatable: "268435456"
 available: "268435456"
 capacity: "268435456"
name: hugepages-2Mi
- allocatable: "269192085504"
 available: "269192085504"
 capacity: "270534262784"
name: memory
type: Node
kind: List
metadata:
 resourceVersion: ""
 selfLink: ""

```

- 1 Each stanza under **zones** describes the resources for a single NUMA zone.
- 2 **resources** describes the current state of the NUMA zone resources. Check that resources listed under **items.zones.resources.available** correspond to the exclusive NUMA zone resources allocated to each guaranteed pod.

### 12.6.1. Reporting more exact resource availability

Enable the **cacheResyncPeriod** specification to help the NUMA Resources Operator report more exact resource availability by monitoring pending resources on nodes and synchronizing this information in the scheduler cache at a defined interval. This also helps to minimize Topology Affinity Error errors because of sub-optimal scheduling decisions. The lower the interval, the greater the network load. The **cacheResyncPeriod** specification is disabled by default.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

## Procedure

1. Delete the currently running **NUMAResourcesScheduler** resource:
  - a. Get the active **NUMAResourcesScheduler** by running the following command:

```
$ oc get NUMAResourcesScheduler
```

### Example output

NAME	AGE
numaresourcesscheduler	92m

- b. Delete the secondary scheduler resource by running the following command:

```
$ oc delete NUMAResourcesScheduler numaresourcesscheduler
```

### Example output

```
numaresourcesscheduler.nodetopology.openshift.io "numaresourcesscheduler" deleted
```

2. Save the following YAML in the file **nro-scheduler-cacheresync.yaml**. This example changes the log level to **Debug**:

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesScheduler
metadata:
 name: numaresourcesscheduler
spec:
 imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-container-rhel8:v4.18"
 cacheResyncPeriod: "5s" 1
```

- 1 Enter an interval value in seconds for synchronization of the scheduler cache. A value of **5s** is typical for most implementations.

3. Create the updated **NUMAResourcesScheduler** resource by running the following command:

```
$ oc create -f nro-scheduler-cacheresync.yaml
```

### Example output

```
numaresourcesscheduler.nodetopology.openshift.io/numaresourcesscheduler created
```

## Verification steps

1. Check that the NUMA-aware scheduler was successfully deployed:

- a. Run the following command to check that the CRD is created successfully:

```
$ oc get crd | grep numaresourcesschedulers
```

#### Example output

NAME	CREATED AT
numaresourcesschedulers.nodetopology.openshift.io	2022-02-25T11:57:03Z

- b. Check that the new custom scheduler is available by running the following command:

```
$ oc get numaresourcesschedulers.nodetopology.openshift.io
```

#### Example output

NAME	AGE
numaresourcesscheduler	3h26m

2. Check that the logs for the scheduler show the increased log level:

- a. Get the list of pods running in the **openshift-numaresources** namespace by running the following command:

```
$ oc get pods -n openshift-numaresources
```

#### Example output

NAME	READY	STATUS	RESTARTS	AGE
numaresources-controller-manager-d87d79587-76mrm	1/1	Running	0	46h
numaresourcesoperator-worker-5wm2k	2/2	Running	0	45h
numaresourcesoperator-worker-pb75c	2/2	Running	0	45h
secondary-scheduler-7976c4d466-qm4sc	1/1	Running	0	21m

- b. Get the logs for the secondary scheduler pod by running the following command:

```
$ oc logs secondary-scheduler-7976c4d466-qm4sc -n openshift-numaresources
```

#### Example output

```
...
I0223 11:04:55.614788 1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
Watch close - *v1.Namespace total 11 items received
I0223 11:04:56.609114 1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
Watch close - *v1.ReplicationController total 10 items received
I0223 11:05:22.626818 1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
Watch close - *v1.StorageClass total 7 items received
I0223 11:05:31.610356 1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
Watch close - *v1.PodDisruptionBudget total 7 items received
I0223 11:05:31.713032 1 eventhandlers.go:186] "Add event for scheduled pod"
pod="openshift-marketplace/certified-operators-thtvq"
I0223 11:05:53.461016 1 eventhandlers.go:244] "Delete event for scheduled pod"
pod="openshift-marketplace/certified-operators-thtvq"
```

## 12.6.2. Changing where high-performance workloads run

The NUMA-aware secondary scheduler is responsible for scheduling high-performance workloads on a worker node and within a NUMA node where the workloads can be optimally processed. By default, the secondary scheduler assigns workloads to the NUMA node within the chosen worker node that has the most available resources.

If you want to change where the workloads run, you can add the **scoringStrategy** setting to the **NUMAResourcesScheduler** custom resource and set its value to either **MostAllocated** or **BalancedAllocation**.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

### Procedure

1. Delete the currently running **NUMAResourcesScheduler** resource by using the following steps:

- a. Get the active **NUMAResourcesScheduler** by running the following command:

```
$ oc get NUMAResourcesScheduler
```

#### Example output

NAME	AGE
numaresourcesscheduler	92m

- b. Delete the secondary scheduler resource by running the following command:

```
$ oc delete NUMAResourcesScheduler numaresourcesscheduler
```

#### Example output

```
numaresourcesscheduler.nodetopology.openshift.io "numaresourcesscheduler" deleted
```

2. Save the following YAML in the file **nro-scheduler-mostallocated.yaml**. This example changes the **scoringStrategy** to **MostAllocated**:

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesScheduler
metadata:
 name: numaresourcesscheduler
spec:
 imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-container-rhel8:v{product-version}"
 scoringStrategy:
 type: "MostAllocated" ①
```

① If the **scoringStrategy** configuration is omitted, the default of **LeastAllocated** applies.

3. Create the updated **NUMAResourcesScheduler** resource by running the following command:

```
$ oc create -f nro-scheduler-mostallocated.yaml
```

#### Example output

```
numaresourcesscheduler.nodetopology.openshift.io/numaresourcesscheduler created
```

### Verification

1. Check that the NUMA-aware scheduler was successfully deployed by using the following steps:

- a. Run the following command to check that the custom resource definition (CRD) is created successfully:

```
$ oc get crd | grep numaresourcesschedulers
```

#### Example output

NAME	CREATED AT
numaresourcesschedulers.nodetopology.openshift.io	2022-02-25T11:57:03Z

- b. Check that the new custom scheduler is available by running the following command:

```
$ oc get numaresourcesschedulers.nodetopology.openshift.io
```

#### Example output

NAME	AGE
numaresourcesscheduler	3h26m

2. Verify that the **ScoringStrategy** has been applied correctly by running the following command to check the relevant **ConfigMap** resource for the scheduler:

```
$ oc get -n openshift-numaresources cm topo-aware-scheduler-config -o yaml | grep scoring -A 1
```

#### Example output

```
scoringStrategy:
type: MostAllocated
```

### 12.6.3. Checking the NUMA-aware scheduler logs

Troubleshoot problems with the NUMA-aware scheduler by reviewing the logs. If required, you can increase the scheduler log level by modifying the **spec.logLevel** field of the **NUMAResourcesScheduler** resource. Acceptable values are **Normal**, **Debug**, and **Trace**, with **Trace** being the most verbose option.



## NOTE

To change the log level of the secondary scheduler, delete the running scheduler resource and re-deploy it with the changed log level. The scheduler is unavailable for scheduling new workloads during this downtime.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

### Procedure

1. Delete the currently running **NUMAResourcesScheduler** resource:
  - a. Get the active **NUMAResourcesScheduler** by running the following command:

```
$ oc get NUMAResourcesScheduler
```

#### Example output

NAME	AGE
numaresourcesscheduler	90m

- b. Delete the secondary scheduler resource by running the following command:

```
$ oc delete NUMAResourcesScheduler numaresourcesscheduler
```

#### Example output

```
numaresourcesscheduler.nodetopology.openshift.io "numaresourcesscheduler" deleted
```

2. Save the following YAML in the file **nro-scheduler-debug.yaml**. This example changes the log level to **Debug**:

```
apiVersion: nodetopology.openshift.io/v1
kind: NUMAResourcesScheduler
metadata:
 name: numaresourcesscheduler
spec:
 imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-container-rhel8:v4.18"
 logLevel: Debug
```

3. Create the updated **Debug** logging **NUMAResourcesScheduler** resource by running the following command:

```
$ oc create -f nro-scheduler-debug.yaml
```

#### Example output

```
numaresourcesscheduler.nodetopology.openshift.io/numaresourcesscheduler created
```

## Verification steps

- Check that the NUMA-aware scheduler was successfully deployed:

- Run the following command to check that the CRD is created successfully:

```
$ oc get crd | grep numaresourcesschedulers
```

### Example output

NAME	CREATED AT
numaresourcesschedulers.nodetopology.openshift.io	2022-02-25T11:57:03Z

- Check that the new custom scheduler is available by running the following command:

```
$ oc get numaresourcesschedulers.nodetopology.openshift.io
```

### Example output

NAME	AGE
numaresourcesscheduler	3h26m

- Check that the logs for the scheduler shows the increased log level:

- Get the list of pods running in the **openshift-numaresources** namespace by running the following command:

```
$ oc get pods -n openshift-numaresources
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE
numaresources-controller-manager-d87d79587-76mrm	1/1	Running	0	46h
numaresourcesoperator-worker-5wm2k	2/2	Running	0	45h
numaresourcesoperator-worker-pb75c	2/2	Running	0	45h
secondary-scheduler-7976c4d466-qm4sc	1/1	Running	0	21m

- Get the logs for the secondary scheduler pod by running the following command:

```
$ oc logs secondary-scheduler-7976c4d466-qm4sc -n openshift-numaresources
```

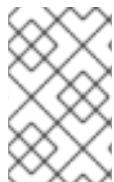
### Example output

```
...
I0223 11:04:55.614788 1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
Watch close - *v1.Namespace total 11 items received
I0223 11:04:56.609114 1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
Watch close - *v1.ReplicationController total 10 items received
I0223 11:05:22.626818 1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
Watch close - *v1.StorageClass total 7 items received
I0223 11:05:31.610356 1 reflector.go:535] k8s.io/client-go/informers/factory.go:134:
```

```
Watch close - *v1.PodDisruptionBudget total 7 items received
I0223 11:05:31.713032 1 eventhandlers.go:186] "Add event for scheduled pod"
pod="openshift-marketplace/certified-operators-thtvq"
I0223 11:05:53.461016 1 eventhandlers.go:244] "Delete event for scheduled pod"
pod="openshift-marketplace/certified-operators-thtvq"
```

## 12.6.4. Troubleshooting the resource topology exporter

Troubleshoot **noderesourcetopologies** objects where unexpected results are occurring by inspecting the corresponding **resource-topology-exporter** logs.



### NOTE

It is recommended that NUMA resource topology exporter instances in the cluster are named for nodes they refer to. For example, a worker node with the name **worker** should have a corresponding **noderesourcetopologies** object called **worker**.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

### Procedure

1. Get the daemonsets managed by the NUMA Resources Operator. Each daemonset has a corresponding **nodeGroup** in the **NUMAResourcesOperator** CR. Run the following command:

```
$ oc get numaresourcesoperators.nodetopology.openshift.io numaresourcesoperator -o
jsonpath=".status.daemonsets[0]"
```

### Example output

```
{"name":"numaresourcesoperator-worker","namespace":"openshift-numaresources"}
```

2. Get the label for the daemonset of interest using the value for **name** from the previous step:

```
$ oc get ds -n openshift-numaresources numaresourcesoperator-worker -o jsonpath="
{.spec.selector.matchLabels}"
```

### Example output

```
{"name":"resource-topology"}
```

3. Get the pods using the **resource-topology** label by running the following command:

```
$ oc get pods -n openshift-numaresources -l name=resource-topology -o wide
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
numaresourcesoperator-worker-5wm2k	2/2	Running	0	2d1h	10.135.0.64	

```
compute-0.example.com
numaresourcesoperator-worker-pb75c 2/2 Running 0 2d1h 10.132.2.33
compute-1.example.com
```

4. Examine the logs of the **resource-topology-exporter** container running on the worker pod that corresponds to the node you are troubleshooting. Run the following command:

```
$ oc logs -n openshift-numaresources -c resource-topology-exporter numaresourcesoperator-worker-pb75c
```

#### Example output

```
I0221 13:38:18.334140 1 main.go:206] using sysinfo:
reservedCpus: 0,1
reservedMemory:
"0": 1178599424
I0221 13:38:18.334370 1 main.go:67] === System information ===
I0221 13:38:18.334381 1 sysinfo.go:231] cpus: reserved "0-1"
I0221 13:38:18.334493 1 sysinfo.go:237] cpus: online "0-103"
I0221 13:38:18.546750 1 main.go:72]
cpus: allocatable "2-103"
hugepages-1Gi:
 numa cell 0 -> 6
 numa cell 1 -> 1
hugepages-2Mi:
 numa cell 0 -> 64
 numa cell 1 -> 128
memory:
 numa cell 0 -> 45758Mi
 numa cell 1 -> 48372Mi
```

#### 12.6.5. Correcting a missing resource topology exporter config map

If you install the NUMA Resources Operator in a cluster with misconfigured cluster settings, in some circumstances, the Operator is shown as active but the logs of the resource topology exporter (RTE) daemon set pods show that the configuration for the RTE is missing, for example:

```
Info: couldn't find configuration in "/etc/resource-topology-exporter/config.yaml"
```

This log message indicates that the **kubeletconfig** with the required configuration was not properly applied in the cluster, resulting in a missing RTE **configmap**. For example, the following cluster is missing a **numaresourcesoperator-worker configmap** custom resource (CR):

```
$ oc get configmap
```

#### Example output

NAME	DATA	AGE
0e2a6bd3.openshift-kni.io	0	6d21h
kube-root-ca.crt	1	6d21h
openshift-service-ca.crt	1	6d21h
topo-aware-scheduler-config	1	6d18h

In a correctly configured cluster, `oc get configmap` also returns a **numaresourcesoperator-worker configmap** CR.

## Prerequisites

- Install the OpenShift Container Platform CLI (**oc**).
- Log in as a user with cluster-admin privileges.
- Install the NUMA Resources Operator and deploy the NUMA-aware secondary scheduler.

## Procedure

1. Compare the values for **spec.machineConfigPoolSelector.matchLabels** in **kubeletconfig** and **metadata.labels** in the **MachineConfigPool** (**mcp**) worker CR using the following commands:
  - a. Check the **kubeletconfig** labels by running the following command:

```
$ oc get kubeletconfig -o yaml
```

### Example output

```
machineConfigPoolSelector:
 matchLabels:
 cnf-worker-tuning: enabled
```

- b. Check the **mcp** labels by running the following command:

```
$ oc get mcp worker -o yaml
```

### Example output

```
labels:
 machineconfiguration.openshift.io/mco-built-in: ""
 pools.operator.machineconfiguration.openshift.io/worker: ""
```

The **cnf-worker-tuning: enabled** label is not present in the **MachineConfigPool** object.

2. Edit the **MachineConfigPool** CR to include the missing label, for example:

```
$ oc edit mcp worker -o yaml
```

### Example output

```
labels:
 machineconfiguration.openshift.io/mco-built-in: ""
 pools.operator.machineconfiguration.openshift.io/worker: ""
 cnf-worker-tuning: enabled
```

3. Apply the label changes and wait for the cluster to apply the updated configuration. Run the following command:

## Verification

- Check that the missing **numaresourcesoperator-worker configmap** CR is applied:

```
$ oc get configmap
```

#### Example output

NAME	DATA	AGE
0e2a6bd3.openshift-kni.io	0	6d21h
kube-root-ca.crt	1	6d21h
numaresourcesoperator-worker	1	5m
openshift-service-ca.crt	1	6d21h
topo-aware-scheduler-config	1	6d18h

### 12.6.6. Collecting NUMA Resources Operator data

You can use the **oc adm must-gather** CLI command to collect information about your cluster, including features and objects associated with the NUMA Resources Operator.

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).

#### Procedure

- To collect NUMA Resources Operator data with **must-gather**, you must specify the NUMA Resources Operator **must-gather** image.

```
$ oc adm must-gather --image=registry.redhat.io/openshift4/numaresources-must-gather-rhel9:v4.18
```

# CHAPTER 13. SCALABILITY AND PERFORMANCE OPTIMIZATION

## 13.1. OPTIMIZING STORAGE

Optimizing storage helps to minimize storage use across all resources. By optimizing storage, administrators help ensure that existing storage resources are working in an efficient manner.

### 13.1.1. Available persistent storage options

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

**Table 13.1. Available storage options**

Storage type	Description	Examples
Block	<ul style="list-style-type: none"> <li>Presented to the operating system (OS) as a block device</li> <li>Suitable for applications that need full control of storage and operate at a low level on files bypassing the file system</li> <li>Also referred to as a Storage Area Network (SAN)</li> <li>Non-shareable, which means that only one client at a time can mount an endpoint of this type</li> </ul>	AWS EBS and VMware vSphere support dynamic persistent volume (PV) provisioning natively in the OpenShift Container Platform.
File	<ul style="list-style-type: none"> <li>Presented to the OS as a file system export to be mounted</li> <li>Also referred to as Network Attached Storage (NAS)</li> <li>Concurrency, latency, file locking mechanisms, and other capabilities vary widely between protocols, implementations, vendors, and scales.</li> </ul>	RHEL NFS, NetApp NFS <sup>[1]</sup> , and Vendor NFS
Object	<ul style="list-style-type: none"> <li>Accessible through a REST API endpoint</li> <li>Configurable for use in the OpenShift image registry</li> <li>Applications must build their drivers into the application and/or container.</li> </ul>	AWS S3

1. NetApp NFS supports dynamic PV provisioning when using the Trident plugin.

### 13.1.2. Recommended configurable storage technology

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

**Table 13.2. Recommended and configurable storage technology**

Storage type	Block	File	Object
ROX <sup>1</sup>	Yes <sup>4</sup>	Yes <sup>4</sup>	Yes
RWX <sup>2</sup>	No	Yes	Yes
Registry	Configurable	Configurable	Recommended
Scaled registry	Not configurable	Configurable	Recommended
Metrics <sup>3</sup>	Recommended	Configurable <sup>5</sup>	Not configurable
Elasticsearch Logging	Recommended	Configurable <sup>6</sup>	Not supported <sup>6</sup>
Loki Logging	Not configurable	Not configurable	Recommended
Apps	Recommended	Recommended	Not configurable <sup>7</sup>

<sup>1</sup> **ReadOnlyMany**

<sup>2</sup> **ReadWriteMany**

<sup>3</sup> Prometheus is the underlying technology used for metrics.

<sup>4</sup> This does not apply to physical disk, VM physical disk, VMDK, loopback over NFS, AWS EBS, and Azure Disk.

<sup>5</sup> For metrics, using file storage with the **ReadWriteMany** (RWX) access mode is unreliable. If you use file storage, do not configure the RWX access mode on any persistent volume claims (PVCs) that are configured for use with metrics.

<sup>6</sup> For logging, review the recommended storage solution in Configuring persistent storage for the log store section. Using NFS storage as a persistent volume or through NAS, such as Gluster, can corrupt the data. Hence, NFS is not supported for Elasticsearch storage and LokiStack log store in OpenShift Container Platform Logging. You must use one persistent volume type per log store.

<sup>7</sup> Object storage is not consumed through OpenShift Container Platform's PVs or PVCs. Apps must integrate with the object storage REST API.

**NOTE**

A scaled registry is an OpenShift image registry where two or more pod replicas are running.

### 13.1.2.1. Specific application storage recommendations

**IMPORTANT**

Testing shows issues with using the NFS server on Red Hat Enterprise Linux (RHEL) as a storage backend for core services. This includes the OpenShift Container Registry and Quay, Prometheus for monitoring storage, and Elasticsearch for logging storage. Therefore, using RHEL NFS to back PVs used by core services is not recommended.

Other NFS implementations in the marketplace might not have these issues. Contact the individual NFS implementation vendor for more information on any testing that was possibly completed against these OpenShift Container Platform core components.

#### 13.1.2.1.1. Registry

In a non-scaled/high-availability (HA) OpenShift image registry cluster deployment:

- The storage technology does not have to support RWX access mode.
- The storage technology must ensure read-after-write consistency.
- The preferred storage technology is object storage followed by block storage.
- File storage is not recommended for OpenShift image registry cluster deployment with production workloads.

#### 13.1.2.1.2. Scaled registry

In a scaled/HA OpenShift image registry cluster deployment:

- The storage technology must support RWX access mode.
- The storage technology must ensure read-after-write consistency.
- The preferred storage technology is object storage.
- Red Hat OpenShift Data Foundation (ODF), Amazon Simple Storage Service (Amazon S3), Google Cloud Storage (GCS), Microsoft Azure Blob Storage, and OpenStack Swift are supported.
- Object storage should be S3 or Swift compliant.
- For non-cloud platforms, such as vSphere and bare metal installations, the only configurable technology is file storage.
- Block storage is not configurable.
- The use of Network File System (NFS) storage with OpenShift Container Platform is supported. However, the use of NFS storage with a scaled registry can cause known issues. For more information, see the Red Hat Knowledgebase solution, [Is NFS supported for OpenShift cluster internal components in Production?](#).

### 13.1.2.1.3. Metrics

In an OpenShift Container Platform hosted metrics cluster deployment:

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



#### IMPORTANT

It is not recommended to use file storage for a hosted metrics cluster deployment with production workloads.

### 13.1.2.1.4. Logging

In an OpenShift Container Platform hosted logging cluster deployment:

- Loki Operator:
  - The preferred storage technology is S3 compatible Object storage.
  - Block storage is not configurable.
- OpenShift Elasticsearch Operator:
  - The preferred storage technology is block storage.
  - Object storage is not supported.



#### NOTE

As of logging version 5.4.3 the OpenShift Elasticsearch Operator is deprecated and is planned to be removed in a future release. Red Hat will provide bug fixes and support for this feature during the current release lifecycle, but this feature will no longer receive enhancements and will be removed. As an alternative to using the OpenShift Elasticsearch Operator to manage the default log storage, you can use the Loki Operator.

### 13.1.2.1.5. Applications

Application use cases vary from application to application, as described in the following examples:

- Storage technologies that support dynamic PV provisioning have low mount time latencies, and are not tied to nodes to support a healthy cluster.
- Application developers are responsible for knowing and understanding the storage requirements for their application, and how it works with the provided storage to ensure that issues do not occur when an application scales or interacts with the storage layer.

### 13.1.2.2. Other specific application storage recommendations



## IMPORTANT

It is not recommended to use RAID configurations on **Write** intensive workloads, such as **etcd**. If you are running **etcd** with a RAID configuration, you might be at risk of encountering performance issues with your workloads.

- Red Hat OpenStack Platform (RHOSP) Cinder: RHOSP Cinder tends to be adept in ROX access mode use cases.
- Databases: Databases (RDBMSs, NoSQL DBs, etc.) tend to perform best with dedicated block storage.
- The etcd database must have enough storage and adequate performance capacity to enable a large cluster. Information about monitoring and benchmarking tools to establish ample storage and a high-performance environment is described in *Recommended etcd practices*.

### 13.1.3. Data storage management

The following table summarizes the main directories that OpenShift Container Platform components write data to.

**Table 13.3. Main directories for storing OpenShift Container Platform data**

Directory	Notes	Sizing	Expected growth
/var/lib/etcd	Used for etcd storage when storing the database.	Less than 20 GB. Database can grow up to 8 GB.	Will grow slowly with the environment. Only storing metadata.  Additional 20-25 GB for every additional 8 GB of memory.
/var/lib/containers	This is the mount point for the CRI-O runtime. Storage used for active container runtimes, including pods, and storage of local images. Not used for registry storage.	50 GB for a node with 16 GB memory. Note that this sizing should not be used to determine minimum cluster requirements.  Additional 20-25 GB for every additional 8 GB of memory.	Growth is limited by capacity for running containers.
/var/log	Log files for all components.	10 to 30 GB.	Log files can grow quickly; size can be managed by growing disks or by using log rotate.

Directory	Notes	Sizing	Expected growth
/var/lib/kubelet	Ephemeral volume storage for pods. This includes anything external that is mounted into a container at runtime. Includes environment variables, kube secrets, and data volumes not backed by persistent volumes.	Varies	Minimal if pods requiring storage are using persistent volumes. If using ephemeral storage, this can grow quickly.
/var/log	<b>Log files for all components.</b>	10 to 30 GB.	<b>Log files can grow quickly; size can be managed by growing disks or by using log rotate.</b>

### 13.1.4. Optimizing storage performance for Microsoft Azure

OpenShift Container Platform and Kubernetes are sensitive to disk performance, and faster storage is recommended, particularly for etcd on the control plane nodes.

For production Azure clusters and clusters with intensive workloads, the virtual machine operating system disk for control plane machines should be able to sustain a tested and recommended minimum throughput of 5000 IOPS / 200MBps. This throughput can be provided by having a minimum of 1 TiB Premium SSD (P30). In Azure and Azure Stack Hub, disk performance is directly dependent on SSD disk sizes. To achieve the throughput supported by a **Standard\_D8s\_v3** virtual machine, or other similar machine types, and the target of 5000 IOPS, at least a P30 disk is required.

Host caching must be set to **ReadOnly** for low latency and high IOPS and throughput when reading data. Reading data from the cache, which is present either in the VM memory or in the local SSD disk, is much faster than reading from the disk, which is in the blob storage.

## 13.2. OPTIMIZING ROUTING

The OpenShift Container Platform HAProxy router can be scaled or configured to optimize performance.

### 13.2.1. Baseline Ingress Controller (router) performance

The OpenShift Container Platform Ingress Controller, or router, is the ingress point for ingress traffic for applications and services that are configured using routes and ingresses.

When evaluating a single HAProxy router performance in terms of HTTP requests handled per second, the performance varies depending on many factors. In particular:

- HTTP keep-alive/close mode
- Route type

- TLS session resumption client support
- Number of concurrent connections per target route
- Number of target routes
- Back end server page size
- Underlying infrastructure (network, CPU, and so on)

While performance in your specific environment will vary, Red Hat lab tests on a public cloud instance of size 4 vCPU/16GB RAM. A single HAProxy router handling 100 routes terminated by backends serving 1kB static pages is able to handle the following number of transactions per second.

In HTTP keep-alive mode scenarios:

Encryption	LoadBalancerService	HostNetwork
none	21515	29622
edge	16743	22913
passthrough	36786	53295
re-encrypt	21583	25198

In HTTP close (no keep-alive) scenarios:

Encryption	LoadBalancerService	HostNetwork
none	5719	8273
edge	2729	4069
passthrough	4121	5344
re-encrypt	2320	2941

The default Ingress Controller configuration was used with the **spec.tuningOptions.threadCount** field set to **4**. Two different endpoint publishing strategies were tested: Load Balancer Service and Host Network. TLS session resumption was used for encrypted routes. With HTTP keep-alive, a single HAProxy router is capable of saturating a 1 Gbit NIC at page sizes as small as 8 kB.

When running on bare metal with modern processors, you can expect roughly twice the performance of the public cloud instance above. This overhead is introduced by the virtualization layer in place on public clouds and holds mostly true for private cloud-based virtualization as well. The following table is a guide to how many applications to use behind the router:

Number of applications	Application type
5-10	static file/web server or caching proxy
100-1000	applications generating dynamic content

In general, HAProxy can support routes for up to 1000 applications, depending on the technology in use. Ingress Controller performance might be limited by the capabilities and performance of the applications behind it, such as language or static versus dynamic content.

Ingress, or router, sharding should be used to serve more routes towards applications and help horizontally scale the routing tier.

For more information on Ingress sharding, see [Configuring Ingress Controller sharding by using route labels](#) and [Configuring Ingress Controller sharding by using namespace labels](#).

You can modify the Ingress Controller deployment by using the information provided in [Setting Ingress Controller thread count](#) for threads and [Ingress Controller configuration parameters](#) for timeouts, and other tuning configurations in the Ingress Controller specification.

### 13.2.2. Configuring Ingress Controller liveness, readiness, and startup probes

Cluster administrators can configure the timeout values for the kubelet's liveness, readiness, and startup probes for router deployments that are managed by the OpenShift Container Platform Ingress Controller (router). The liveness and readiness probes of the router use the default timeout value of 1 second, which is too brief when networking or runtime performance is severely degraded. Probe timeouts can cause unwanted router restarts that interrupt application connections. The ability to set larger timeout values can reduce the risk of unnecessary and unwanted restarts.

You can update the **timeoutSeconds** value on the **livenessProbe**, **readinessProbe**, and **startupProbe** parameters of the router container.

Parameter	Description
<b>livenessProbe</b>	The <b>livenessProbe</b> reports to the kubelet whether a pod is dead and needs to be restarted.
<b>readinessProbe</b>	The <b>readinessProbe</b> reports whether a pod is healthy or unhealthy. When the readiness probe reports an unhealthy pod, then the kubelet marks the pod as not ready to accept traffic. Subsequently, the endpoints for that pod are marked as not ready, and this status propagates to the kube-proxy. On cloud platforms with a configured load balancer, the kube-proxy communicates to the cloud load-balancer not to send traffic to the node with that pod.
<b>startupProbe</b>	The <b>startupProbe</b> gives the router pod up to 2 minutes to initialize before the kubelet begins sending the router liveness and readiness probes. This initialization time can prevent routers with many routes or endpoints from prematurely restarting.



## IMPORTANT

The timeout configuration option is an advanced tuning technique that can be used to work around issues. However, these issues should eventually be diagnosed and possibly a support case or [Jira issue](#) opened for any issues that causes probes to time out.

The following example demonstrates how you can directly patch the default router deployment to set a 5-second timeout for the liveness and readiness probes:

```
$ oc -n openshift-ingress patch deploy/router-default --type=strategic --patch='{"spec":{"template":{"spec":{"containers":[{"name":"router","livenessProbe":{"timeoutSeconds":5}, "readinessProbe":{"timeoutSeconds":5}}]}}}'
```

## Verification

```
$ oc -n openshift-ingress describe deploy/router-default | grep -e Liveness: -e Readiness:
Liveness: http-get http://:1936/healthz delay=0s timeout=5s period=10s #success=1 #failure=3
Readiness: http-get http://:1936/healthz/ready delay=0s timeout=5s period=10s #success=1
#failure=3
```

### 13.2.3. Configuring HAProxy reload interval

When you update a route or an endpoint associated with a route, the OpenShift Container Platform router updates the configuration for HAProxy. Then, HAProxy reloads the updated configuration for those changes to take effect. When HAProxy reloads, it generates a new process that handles new connections using the updated configuration.

HAProxy keeps the old process running to handle existing connections until those connections are all closed. When old processes have long-lived connections, these processes can accumulate and consume resources.

The default minimum HAProxy reload interval is five seconds. You can configure an Ingress Controller using its **spec.tuningOptions.reloadInterval** field to set a longer minimum reload interval.



## WARNING

Setting a large value for the minimum HAProxy reload interval can cause latency in observing updates to routes and their endpoints. To lessen the risk, avoid setting a value larger than the tolerable latency for updates.

## Procedure

- Change the minimum HAProxy reload interval of the default Ingress Controller to 15 seconds by running the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --
patch='{"spec":{"tuningOptions":{"reloadInterval":"15s"}}}'
```

## 13.3. OPTIMIZING NETWORKING

[OVN-Kubernetes](#) uses Generic Network Virtualization Encapsulation (Geneve) a protocol similar to Geneve to tunnel traffic between nodes. This network can be tuned by using network interface controller (NIC) offloads.

Geneve provides benefits over VLANs, such as an increase in networks from 4096 to over 16 million, and layer 2 connectivity across physical networks. This allows for all pods behind a service to communicate with each other, even if they are running on different systems.

Cloud, virtual, and bare-metal environments running OpenShift Container Platform can use a high percentage of a NIC's capabilities with minimal tuning. Production clusters using OVN-Kubernetes with Geneve tunneling can handle high-throughput traffic effectively and scale up (for example, utilizing 100 Gbps NICs) and scale out (for example, adding more NICs) without requiring special configuration.

In some high-performance scenarios where maximum efficiency is critical, targeted performance tuning can help optimize CPU usage, reduce overhead, and ensure that you are making full use of the NIC's capabilities.

For environments where maximum throughput and CPU efficiency are critical, you can further optimize performance with the following strategies:

- Validate network performance using tools such as **iPerf3** and **k8s-netperf**. These tools allow you to benchmark throughput, latency, and packets-per-second (PPS) across pod and node interfaces.
- Evaluate OVN-Kubernetes User Defined Networking (UDN) routing techniques, such as border gateway protocol (BGP).
- Use Geneve-offload capable network adapters. Geneve-offload moves the packet checksum calculation and associated CPU overhead off of the system CPU and onto dedicated hardware on the network adapter. This frees up CPU cycles for use by pods and applications, and allows users to use the full bandwidth of their network infrastructure.

### 13.3.1. Optimizing the MTU for your network

There are two important maximum transmission units (MTUs): the network interface controller (NIC) MTU and the cluster network MTU.

The NIC MTU is configured at the time of OpenShift Container Platform installation, and you can also change the MTU of a cluster as a postinstallation task. For more information, see "Changing cluster network MTU".

For a cluster that uses the OVN-Kubernetes plugin, the MTU must be less than **100** bytes to the maximum supported value of the NIC of your network. If you are optimizing for throughput, choose the largest possible value, such as **8900**. If you are optimizing for lowest latency, choose a lower value.



#### IMPORTANT

If your cluster uses the OVN-Kubernetes plugin and the network uses a NIC to send and receive unfragmented jumbo frame packets over the network, you must specify **9000** bytes as the MTU value for the NIC so that pods do not fail.

#### Additional resources

- [Changing cluster network MTU](#)

### 13.3.2. Recommended practices for installing large scale clusters

When installing large clusters or scaling the cluster to larger node counts, set the cluster network **cidr** accordingly in your **install-config.yaml** file before you install the cluster.

#### Example **install-config.yaml** file with a network configuration for a cluster with a large node count

```
networking:
 clusterNetwork:
 - cidr: 10.128.0.0/14
 hostPrefix: 23
 machineNetwork:
 - cidr: 10.0.0.0/16
 networkType: OVNKubernetes
 serviceNetwork:
 - 172.30.0.0/16
```

The default cluster network **cidr 10.128.0.0/14** cannot be used if the cluster size is more than 500 nodes. The **cidr** must be set to **10.128.0.0/12** or **10.128.0.0/10** to get to larger node counts beyond 500 nodes.

### 13.3.3. Impact of IPsec

Because encrypting and decrypting node hosts uses CPU power, performance is affected both in throughput and CPU usage on the nodes when encryption is enabled, regardless of the IP security system being used.

IPSec encrypts traffic at the IP payload level, before it hits the NIC, protecting fields that would otherwise be used for NIC offloading. This means that some NIC acceleration features might not be usable when IPSec is enabled and leads to decreased throughput and increased CPU usage.

### 13.3.4. Additional resources

- [Specifying advanced network configuration](#)
- [Cluster Network Operator configuration](#)
- [Improving cluster stability in high latency environments using worker latency profiles](#)

## 13.4. OPTIMIZING CPU USAGE WITH MOUNT NAMESPACE ENCAPSULATION

You can optimize CPU usage in OpenShift Container Platform clusters by using mount namespace encapsulation to provide a private namespace for kubelet and CRI-O processes. This reduces the cluster CPU resources used by systemd with no difference in functionality.



## IMPORTANT

Mount namespace encapsulation 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](#).

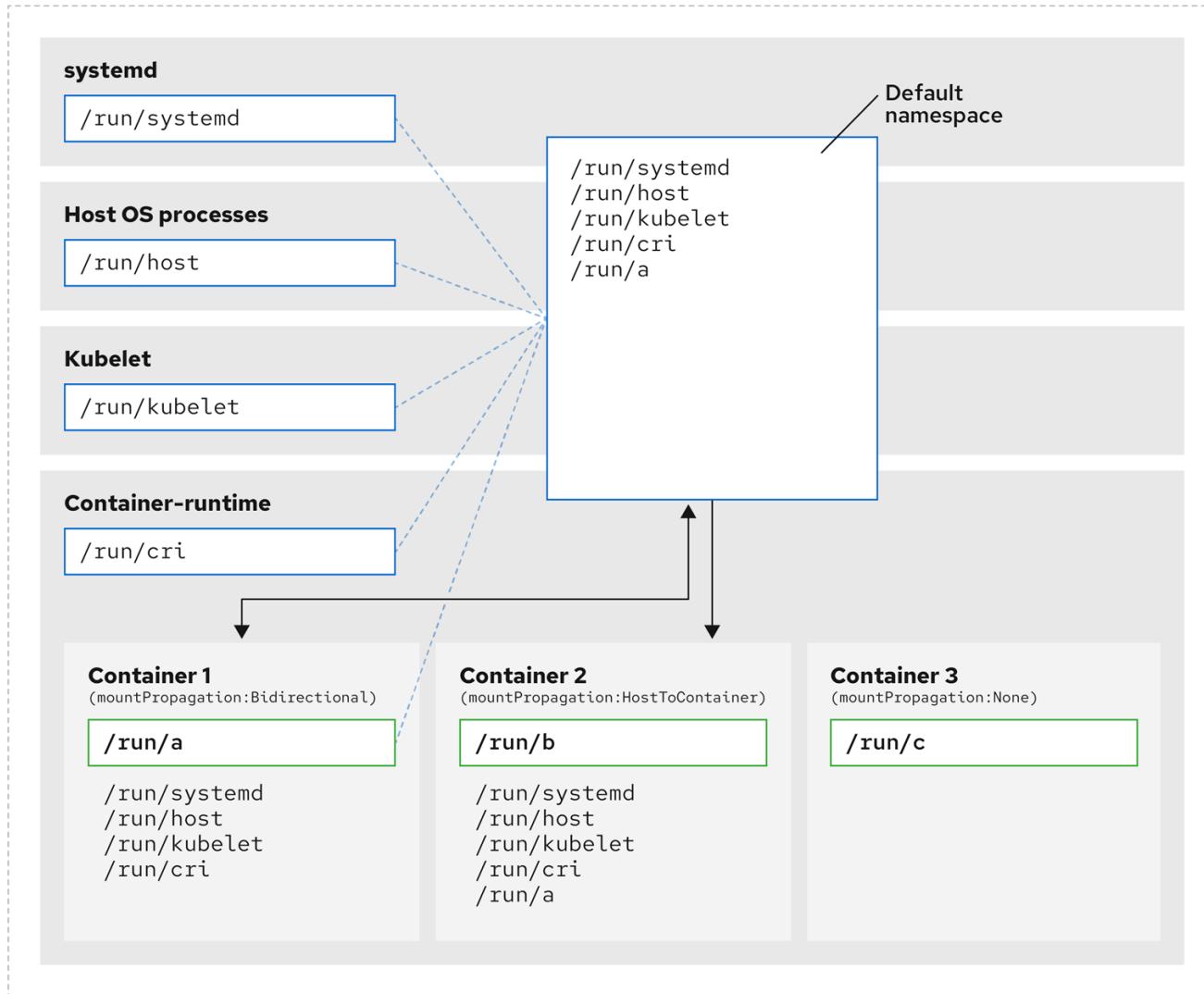
### 13.4.1. Encapsulating mount namespaces

Mount namespaces are used to isolate mount points so that processes in different namespaces cannot view each others' files. Encapsulation is the process of moving Kubernetes mount namespaces to an alternative location where they will not be constantly scanned by the host operating system.

The host operating system uses systemd to constantly scan all mount namespaces: both the standard Linux mounts and the numerous mounts that Kubernetes uses to operate. The current implementation of kubelet and CRI-O both use the top-level namespace for all container runtime and kubelet mount points. However, encapsulating these container-specific mount points in a private namespace reduces systemd overhead with no difference in functionality. Using a separate mount namespace for both CRI-O and kubelet can encapsulate container-specific mounts from any systemd or other host operating system interaction.

This ability to potentially achieve major CPU optimization is now available to all OpenShift Container Platform administrators. Encapsulation can also improve security by storing Kubernetes-specific mount points in a location safe from inspection by unprivileged users.

The following diagrams illustrate a Kubernetes installation before and after encapsulation. Both scenarios show example containers which have mount propagation settings of bidirectional, host-to-container, and none.

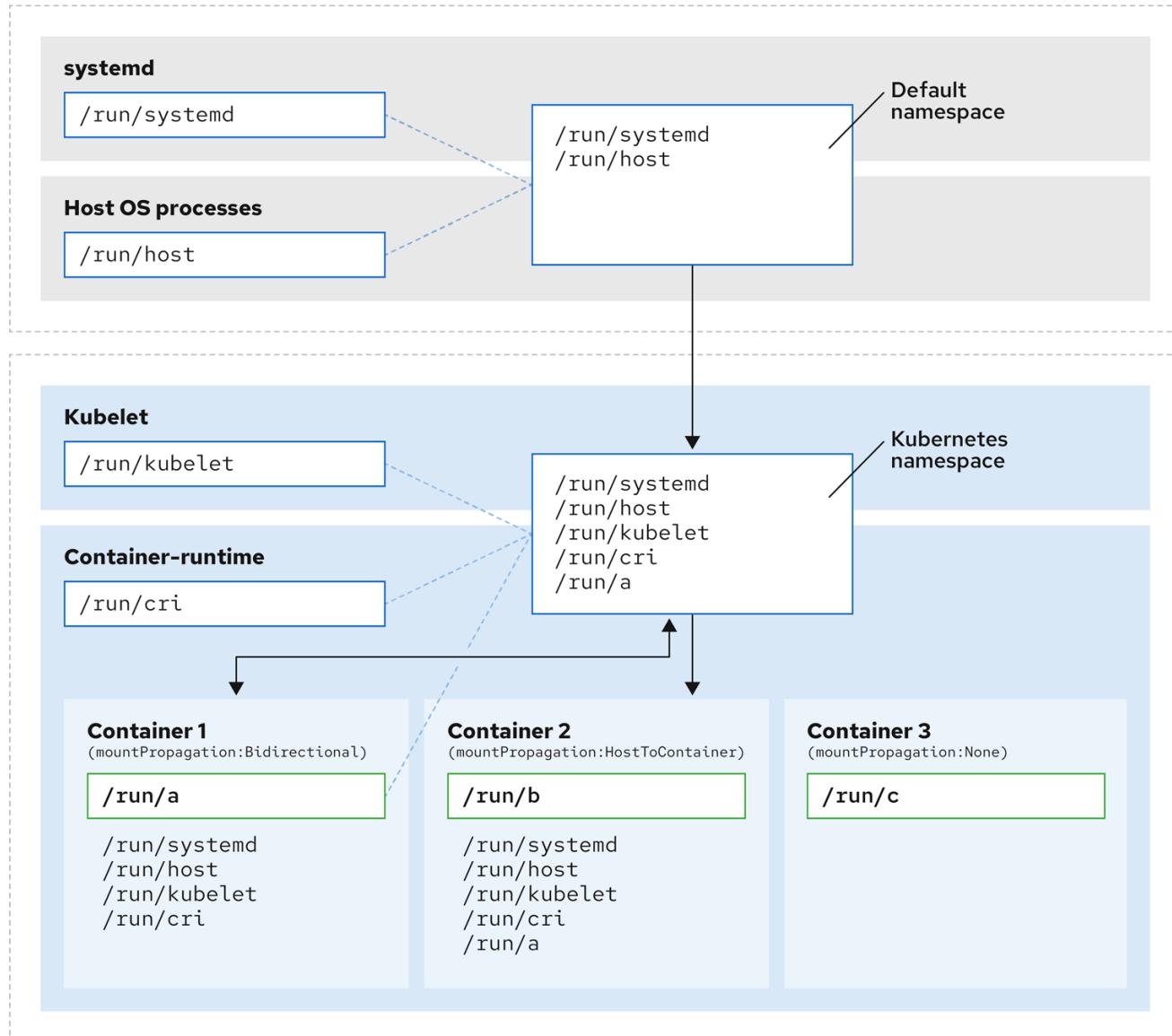


290\_OpenShift\_1122

Here we see systemd, host operating system processes, kubelet, and the container runtime sharing a single mount namespace.

- systemd, host operating system processes, kubelet, and the container runtime each have access to and visibility of all mount points.
- Container 1, configured with bidirectional mount propagation, can access systemd and host mounts, kubelet and CRI-O mounts. A mount originating in Container 1, such as `/run/a` is visible to systemd, host operating system processes, kubelet, container runtime, and other containers with host-to-container or bidirectional mount propagation configured (as in Container 2).
- Container 2, configured with host-to-container mount propagation, can access systemd and host mounts, kubelet and CRI-O mounts. A mount originating in Container 2, such as `/run/b`, is not visible to any other context.
- Container 3, configured with no mount propagation, has no visibility of external mount points. A mount originating in Container 3, such as `/run/c`, is not visible to any other context.

The following diagram illustrates the system state after encapsulation.



290\_OpenShift\_1122

- The main systemd process is no longer devoted to unnecessary scanning of Kubernetes-specific mount points. It only monitors systemd-specific and host mount points.
- The host operating system processes can access only the systemd and host mount points.
- Using a separate mount namespace for both CRI-O and kubelet completely separates all container-specific mounts away from any systemd or other host operating system interaction whatsoever.
- The behavior of Container 1 is unchanged, except a mount it creates such as `/run/a` is no longer visible to systemd or host operating system processes. It is still visible to kubelet, CRI-O, and other containers with host-to-container or bidirectional mount propagation configured (like Container 2).
- The behavior of Container 2 and Container 3 is unchanged.

### 13.4.2. Configuring mount namespace encapsulation

You can configure mount namespace encapsulation so that a cluster runs with less resource overhead.



## NOTE

Mount namespace encapsulation is a Technology Preview feature and it is disabled by default. To use it, you must enable the feature manually.

### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.

### Procedure

1. Create a file called **mount\_namespace\_config.yaml** with the following YAML:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
 labels:
 machineconfiguration.openshift.io/role: master
 name: 99-kubens-master
spec:
 config:
 ignition:
 version: 3.2.0
 systemd:
 units:
 - enabled: true
 name: kubens.service

apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
 labels:
 machineconfiguration.openshift.io/role: worker
 name: 99-kubens-worker
spec:
 config:
 ignition:
 version: 3.2.0
 systemd:
 units:
 - enabled: true
 name: kubens.service
```

2. Apply the mount namespace **MachineConfig** CR by running the following command:

```
$ oc apply -f mount_namespace_config.yaml
```

### Example output

```
machineconfig.machineconfiguration.openshift.io/99-kubens-master created
machineconfig.machineconfiguration.openshift.io/99-kubens-worker created
```

- The **MachineConfig** CR can take up to 30 minutes to finish being applied in the cluster. You can check the status of the **MachineConfig** CR by running the following command:

```
$ oc get mcp
```

#### Example output

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT		
DEGRADED MACHINECOUNT	AGE			
master	rendered-master-03d4bc4befb0f4ed3566a2c8f7636751	False	True	False
3	0	0	45m	
worker	rendered-worker-10577f6ab0117ed1825f8af2ac687ddf	False	True	False
3	1	1		

- Wait for the **MachineConfig** CR to be applied successfully across all control plane and worker nodes after running the following command:

```
$ oc wait --for=condition=Updated mcp --all --timeout=30m
```

#### Example output

```
machineconfigpool.machineconfiguration.openshift.io/master condition met
machineconfigpool.machineconfiguration.openshift.io/worker condition met
```

### Verification

To verify encapsulation for a cluster host, run the following commands:

- Open a debug shell to the cluster host:

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

- Open a **chroot** session:

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

- Check the systemd mount namespace:

```
sh-4.4# readlink /proc/1/ns/mnt
```

#### Example output

```
mnt:[4026531953]
```

- Check kubelet mount namespace:

```
sh-4.4# readlink /proc/$(pgrep kubelet)/ns/mnt
```

#### Example output

```
mnt:[4026531840]
```

5. Check the CRI-O mount namespace:

```
sh-4.4# readlink /proc/$(pgrep crio)/ns/mnt
```

#### Example output

```
mnt:[4026531840]
```

These commands return the mount namespaces associated with systemd, kubelet, and the container runtime. In OpenShift Container Platform, the container runtime is CRI-O.

Encapsulation is in effect if systemd is in a different mount namespace to kubelet and CRI-O as in the above example. Encapsulation is not in effect if all three processes are in the same mount namespace.

### 13.4.3. Inspecting encapsulated namespaces

You can inspect Kubernetes-specific mount points in the cluster host operating system for debugging or auditing purposes by using the **kubensenter** script that is available in Red Hat Enterprise Linux CoreOS (RHCOS).

SSH shell sessions to the cluster host are in the default namespace. To inspect Kubernetes-specific mount points in an SSH shell prompt, you need to run the **kubensenter** script as root. The **kubensenter** script is aware of the state of the mount encapsulation, and is safe to run even if encapsulation is not enabled.



#### NOTE

**oc debug** remote shell sessions start inside the Kubernetes namespace by default. You do not need to run **kubensenter** to inspect mount points when you use **oc debug**.

If the encapsulation feature is not enabled, the **kubensenter findmnt** and **findmnt** commands return the same output, regardless of whether they are run in an **oc debug** session or in an SSH shell prompt.

#### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have configured SSH access to the cluster host.

#### Procedure

1. Open a remote SSH shell to the cluster host. For example:

```
$ ssh core@<node_name>
```

2. Run commands using the provided **kubensenter** script as the root user. To run a single command inside the Kubernetes namespace, provide the command and any arguments to the **kubensenter** script. For example, to run the **findmnt** command inside the Kubernetes namespace, run the following command:

```
[core@control-plane-1 ~]$ sudo kubensenter findmnt
```

#### Example output

```
kubensenter: Autodetect: kubens.service namespace found at /run/kubens/mnt
TARGET SOURCE FSTYPE OPTIONS
/
/dev/sda4[/ostree/deploy/rhcos/deploy/32074f0e8e5ec453e56f5a8a7bc9347eaa4172349ceab9
c22b709d9d71a3f4b0.0]
| xfs
rw,relatime,seclabel,attr2,inode64,logbufs=8,logbsize=32k,prjquota
 shm tmpfs
...
...
```

- To start a new interactive shell inside the Kubernetes namespace, run the **kubensenter** script without any arguments:

```
[core@control-plane-1 ~]$ sudo kubensenter
```

#### Example output

```
kubensenter: Autodetect: kubens.service namespace found at /run/kubens/mnt
```

### 13.4.4. Running additional services in the encapsulated namespace

Any monitoring tool that relies on the ability to run in the host operating system and have visibility of mount points created by kubelet, CRI-O, or containers themselves, must enter the container mount namespace to see these mount points. The **kubensenter** script that is provided with OpenShift Container Platform executes another command inside the Kubernetes mount point and can be used to adapt any existing tools.

The **kubensenter** script is aware of the state of the mount encapsulation feature status, and is safe to run even if encapsulation is not enabled. In that case the script executes the provided command in the default mount namespace.

For example, if a systemd service needs to run inside the new Kubernetes mount namespace, edit the service file and use the **ExecStart=** command line with **kubensenter**.

```
[Unit]
Description=Example service
[Service]
ExecStart=/usr/bin/kubensenter /path/to/original/command arg1 arg2
```

### 13.4.5. Additional resources

- [What are namespaces](#)
- [Manage containers in namespaces by using nsenter](#)
- [MachineConfig](#)

# CHAPTER 14. MANAGING BARE-METAL HOSTS

When you install OpenShift Container Platform on a bare-metal cluster, you can provision and manage bare-metal nodes by using **machine** and **machineset** custom resources (CRs) for bare-metal hosts that exist in the cluster.

## 14.1. ABOUT BARE METAL HOSTS AND NODES

To provision a Red Hat Enterprise Linux CoreOS (RHCOS) bare metal host as a node in your cluster, first create a **MachineSet** custom resource (CR) object that corresponds to the bare metal host hardware. Bare metal host compute machine sets describe infrastructure components specific to your configuration. You apply specific Kubernetes labels to these compute machine sets and then update the infrastructure components to run on only those machines.

**Machine** CR's are created automatically when you scale up the relevant **MachineSet** containing a **metal3.io/autoscale-to-hosts** annotation. OpenShift Container Platform uses **Machine** CR's to provision the bare metal node that corresponds to the host as specified in the **MachineSet** CR.

## 14.2. MAINTAINING BARE METAL HOSTS

You can maintain the details of the bare metal hosts in your cluster from the OpenShift Container Platform web console. Navigate to **Compute → Bare Metal Hosts**, and select a task from the **Actions** drop down menu. Here you can manage items such as BMC details, boot MAC address for the host, enable power management, and so on. You can also review the details of the network interfaces and drives for the host.

You can move a bare metal host into maintenance mode. When you move a host into maintenance mode, the scheduler moves all managed workloads off the corresponding bare metal node. No new workloads are scheduled while in maintenance mode.

You can deprovision a bare metal host in the web console. Deprovisioning a host does the following actions:

1. Annotates the bare metal host CR with **cluster.k8s.io/delete-machine: true**
2. Scales down the related compute machine set



### NOTE

Powering off the host without first moving the daemon set and unmanaged static pods to another node can cause service disruption and loss of data.

### Additional resources

- [Adding compute machines to bare metal](#)

### 14.2.1. Adding a bare metal host to the cluster using the web console

You can add bare metal hosts to the cluster in the web console.

### Prerequisites

- Install an RHCOS cluster on bare metal.

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

### Procedure

1. In the web console, navigate to **Compute → Bare Metal Hosts**
2. Select **Add Host → New with Dialog**.
3. Specify a unique name for the new bare metal host.
4. Set the **Boot MAC address**.
5. Set the **Baseboard Management Console (BMC) Address**
6. Enter the user credentials for the host's baseboard management controller (BMC).
7. Select to power on the host after creation, and select **Create**.
8. Scale up the number of replicas to match the number of available bare metal hosts. Navigate to **Compute → MachineSets**, and increase the number of machine replicas in the cluster by selecting **Edit Machine count** from the **Actions** drop-down menu.



#### NOTE

You can also manage the number of bare metal nodes using the **oc scale** command and the appropriate bare metal compute machine set.

### 14.2.2. Adding a bare metal host to the cluster using YAML in the web console

You can add bare metal hosts to the cluster in the web console using a YAML file that describes the bare metal host.

#### Prerequisites

- Install a RHCOS compute machine on bare metal infrastructure for use in the cluster.
- Log in as a user with **cluster-admin** privileges.
- Create a **Secret** CR for the bare metal host.

#### Procedure

1. In the web console, navigate to **Compute → Bare Metal Hosts**.
2. Select **Add Host → New from YAML**.
3. Copy and paste the below YAML, modifying the relevant fields with the details of your host:

```
apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
 name: <bare_metal_host_name>
spec:
 online: true
 bmc:
```

```
address: <bmc_address>
credentialsName: <secret_credentials_name> ①
disableCertificateVerification: True ②
bootMACAddress: <host_boot_mac_address>
```

- ① **credentialsName** must reference a valid **Secret** CR. The **baremetal-operator** cannot manage the bare metal host without a valid **Secret** referenced in the **credentialsName**. For more information about secrets and how to create them, see [Understanding secrets](#).
- ② Setting **disableCertificateVerification** to **true** disables TLS host validation between the cluster and the baseboard management controller (BMC).

4. Select **Create** to save the YAML and create the new bare metal host.
5. Scale up the number of replicas to match the number of available bare metal hosts. Navigate to **Compute → MachineSets**, and increase the number of machines in the cluster by selecting **Edit Machine count** from the **Actions** drop-down menu.



#### NOTE

You can also manage the number of bare metal nodes using the **oc scale** command and the appropriate bare metal compute machine set.

### 14.2.3. Automatically scaling machines to the number of available bare metal hosts

To automatically create the number of **Machine** objects that matches the number of available **BareMetalHost** objects, add a **metal3.io/autoscale-to-hosts** annotation to the **MachineSet** object.

#### Prerequisites

- Install RHCOS bare metal compute machines for use in the cluster, and create corresponding **BareMetalHost** objects.
- Install the OpenShift Container Platform CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

1. Annotate the compute machine set that you want to configure for automatic scaling by adding the **metal3.io/autoscale-to-hosts** annotation. Replace **<machineset>** with the name of the compute machine set.

```
$ oc annotate machineset <machineset> -n openshift-machine-api 'metal3.io/autoscale-to-hosts=<any_value>'
```

Wait for the new scaled machines to start.

**NOTE**

When you use a **BareMetalHost** object to create a machine in the cluster and labels or selectors are subsequently changed on the **BareMetalHost**, the **BareMetalHost** object continues to be counted against the **MachineSet** that the **Machine** object was created from.

#### 14.2.4. Removing bare metal hosts from the provisioner node

In certain circumstances, you might want to temporarily remove bare metal hosts from the provisioner node. For example, during provisioning when a bare metal host reboot is triggered by using the OpenShift Container Platform administration console or as a result of a Machine Config Pool update, OpenShift Container Platform logs into the integrated Dell Remote Access Controller (iDrac) and issues a delete of the job queue.

To prevent the management of the number of **Machine** objects that matches the number of available **BareMetalHost** objects, add a **baremetalhost.metal3.io/detached** annotation to the **MachineSet** object.

**NOTE**

This annotation has an effect for only **BareMetalHost** objects that are in either **Provisioned**, **ExternallyProvisioned** or **Ready/Available** state.

#### Prerequisites

- Install RHCOS bare metal compute machines for use in the cluster and create corresponding **BareMetalHost** objects.
- Install the OpenShift Container Platform CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

1. Annotate the compute machine set that you want to remove from the provisioner node by adding the **baremetalhost.metal3.io/detached** annotation.

```
$ oc annotate machineset <machineset> -n openshift-machine-api
'baremetalhost.metal3.io/detached'
```

Wait for the new machines to start.

**NOTE**

When you use a **BareMetalHost** object to create a machine in the cluster and labels or selectors are subsequently changed on the **BareMetalHost**, the **BareMetalHost** object continues to be counted against the **MachineSet** that the **Machine** object was created from.

2. In the provisioning use case, remove the annotation after the reboot is complete by using the following command:

```
$ oc annotate machineset <machineset> -n openshift-machine-api
'baremetalhost.metal3.io/detached-'
```

## Additional resources

- [Expanding the cluster](#)
- [MachineHealthChecks on bare metal](#)

### 14.2.5. Powering off bare-metal hosts

You can power off bare-metal cluster hosts in the web console or by applying a patch in the cluster by using the OpenShift CLI (**oc**). Before you power off a host, you should mark the node as unschedulable and drain all pods and workloads from the node.

#### Prerequisites

- You have installed a RHCOS compute machine on bare-metal infrastructure for use in the cluster.
- You have logged in as a user with **cluster-admin** privileges.
- You have configured the host to be managed and have added BMC credentials for the cluster host. You can add BMC credentials by applying a **Secret** custom resource (CR) in the cluster or by logging in to the web console and configuring the bare-metal host to be managed.

#### Procedure

1. In the web console, mark the node that you want to power off as unschedulable. Perform the following steps:
  - a. Navigate to **Nodes** and select the node that you want to power off. Expand the **Actions** menu and select **Mark as unschedulable**.
  - b. Manually delete or relocate running pods on the node by adjusting the pod deployments or scaling down workloads on the node to zero. Wait for the drain process to complete.
  - c. Navigate to **Compute → Bare Metal Hosts**
  - d. Expand the **Options menu** for the bare-metal host that you want to power off, and select **Power Off**. Select **Immediate power off**.
2. Alternatively, you can patch the **BareMetalHost** resource for the host that you want to power off by using **oc**.
  - a. Get the name of the managed bare-metal host. Run the following command:

```
$ oc get baremetalhosts -n openshift-machine-api -o jsonpath='{range .items[*]}
{.metadata.name}{"\t"}{.status.provisioning.state}{"\n"}{end}'
```

#### Example output

```
master-0.example.com managed
master-1.example.com managed
master-2.example.com managed
```

```
worker-0.example.com managed
worker-1.example.com managed
worker-2.example.com managed
```

- b. Mark the node as unschedulable:

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

1 **<bare\_metal\_host>** is the host that you want to shut down, for example, **worker-2.example.com**.

- c. Drain all pods on the node:

```
$ oc adm drain <bare_metal_host> --force=true
```

Pods that are backed by replication controllers are rescheduled to other available nodes in the cluster.

- d. Safely power off the bare-metal host. Run the following command:

```
$ oc patch <bare_metal_host> --type json -p '[{"op": "replace", "path": "/spec/online", "value": false}]'
```

- e. After you power on the host, make the node schedulable for workloads. Run the following command:

```
$ oc adm uncordon <bare_metal_host>
```

# CHAPTER 15. WHAT HUGE PAGES DO AND HOW THEY ARE CONSUMED BY APPLICATIONS

## 15.1. WHAT HUGE PAGES DO

Memory is managed in blocks known as pages. On most systems, a page is 4Ki. 1Mi of memory is equal to 256 pages; 1Gi of memory is 256,000 pages, and so on. CPUs have a built-in memory management unit that manages a list of these pages in hardware. The Translation Lookaside Buffer (TLB) is a small hardware cache of virtual-to-physical page mappings. If the virtual address passed in a hardware instruction can be found in the TLB, the mapping can be determined quickly. If not, a TLB miss occurs, and the system falls back to slower, software-based address translation, resulting in performance issues. Since the size of the TLB is fixed, the only way to reduce the chance of a TLB miss is to increase the page size.

A huge page is a memory page that is larger than 4Ki. On x86\_64 architectures, there are two common huge page sizes: 2Mi and 1Gi. Sizes vary on other architectures. To use huge pages, code must be written so that applications are aware of them. Transparent Huge Pages (THP) attempt to automate the management of huge pages without application knowledge, but they have limitations. In particular, they are limited to 2Mi page sizes. THP can lead to performance degradation on nodes with high memory utilization or fragmentation due to defragmenting efforts of THP, which can lock memory pages. For this reason, some applications may be designed to (or recommend) usage of pre-allocated huge pages instead of THP.

In OpenShift Container Platform, applications in a pod can allocate and consume pre-allocated huge pages.

## 15.2. HOW HUGE PAGES ARE CONSUMED BY APPS

Nodes must pre-allocate huge pages in order for the node to report its huge page capacity. A node can only pre-allocate huge pages for a single size.

Huge pages can be consumed through container-level resource requirements using the resource name **hugepages-<size>**, where size is the most compact binary notation using integer values supported on a particular node. For example, if a node supports 2048KiB page sizes, it exposes a schedulable resource **hugepages-2Mi**. Unlike CPU or memory, huge pages do not support over-commitment.

```
apiVersion: v1
kind: Pod
metadata:
 generateName: hugepages-volume-
spec:
 containers:
 - securityContext:
 privileged: true
 image: rhel7:latest
 command:
 - sleep
 - inf
 name: example
 volumeMounts:
 - mountPath: /dev/hugepages
 name: hugepage
 resources:
 limits:
```

```

hugepages-2Mi: 100Mi ①
memory: "1Gi"
cpu: "1"
volumes:
- name: hugepage
 emptyDir:
 medium: HugePages

```

- ① Specify the amount of memory for **hugepages** as the exact amount to be allocated. Do not specify this value as the amount of memory for **hugepages** multiplied by the size of the page. For example, given a huge page size of 2MB, if you want to use 100MB of huge-page-backed RAM for your application, then you would allocate 50 huge pages. OpenShift Container Platform handles the math for you. As in the above example, you can specify **100MB** directly.

### Allocating huge pages of a specific size

Some platforms support multiple huge page sizes. To allocate huge pages of a specific size, precede the huge pages boot command parameters with a huge page size selection parameter **hugepagesz=<size>**. The **<size>** value must be specified in bytes with an optional scale suffix [**kKmMgG**]. The default huge page size can be defined with the **default\_hugepagesz=<size>** boot parameter.

### Huge page requirements

- Huge page requests must equal the limits. This is the default if limits are specified, but requests are not.
- Huge pages are isolated at a pod scope. Container isolation is planned in a future iteration.
- **EmptyDir** volumes backed by huge pages must not consume more huge page memory than the pod request.
- Applications that consume huge pages via **shmget()** with **SHM\_HUGETLB** must run with a supplemental group that matches *proc/sys/vm/hugetlb\_shm\_group*.

## 15.3. CONSUMING HUGE PAGES RESOURCES USING THE DOWNWARD API

You can use the Downward API to inject information about the huge pages resources that are consumed by a container.

You can inject the resource allocation as environment variables, a volume plugin, or both. Applications that you develop and run in the container can determine the resources that are available by reading the environment variables or files in the specified volumes.

### Procedure

1. Create a **hugepages-volume-pod.yaml** file that is similar to the following example:

```

apiVersion: v1
kind: Pod
metadata:
 generateName: hugepages-volume-
 labels:
 app: hugepages-example

```

```

spec:
 containers:
 - securityContext:
 capabilities:
 add: ["IPC_LOCK"]
 image: rhel7:latest
 command:
 - sleep
 - inf
 name: example
 volumeMounts:
 - mountPath: /dev/hugepages
 name: hugepage
 - mountPath: /etc/podinfo
 name: podinfo
 resources:
 limits:
 hugepages-1Gi: 2Gi
 memory: "1Gi"
 cpu: "1"
 requests:
 hugepages-1Gi: 2Gi
 env:
 - name: REQUESTS_HUGEPAGES_1GI \ ①
 valueFrom:
 resourceFieldRef:
 containerName: example
 resource: requests.hugepages-1Gi
 volumes:
 - name: hugepage
 emptyDir:
 medium: HugePages
 - name: podinfo
 downwardAPI:
 items:
 - path: "hugepages_1G_request" \ ②
 resourceFieldRef:
 containerName: example
 resource: requests.hugepages-1Gi
 divisor: 1Gi

```

- ① Specifies to read the resource use from **requests.hugepages-1Gi** and expose the value as the **REQUESTS\_HUGEPAGES\_1GI** environment variable.
- ② Specifies to read the resource use from **requests.hugepages-1Gi** and expose the value as the file **/etc/podinfo/hugepages\_1G\_request**.

## 2. Create the pod from the **hugepages-volume-pod.yaml** file:

```
$ oc create -f hugepages-volume-pod.yaml
```

### Verification

1. Check the value of the **REQUESTS\_HUGEPAGES\_1GI** environment variable:

```
$ oc exec -it $(oc get pods -l app=hugepages-example -o
jsonpath='{.items[0].metadata.name}') \
-- env | grep REQUESTS_HUGEPAGES_1Gi
```

### Example output

```
REQUESTS_HUGEPAGES_1Gi=2147483648
```

- Check the value of the **/etc/podinfo/hugepages\_1G\_request** file:

```
$ oc exec -it $(oc get pods -l app=hugepages-example -o
jsonpath='{.items[0].metadata.name}') \
-- cat /etc/podinfo/hugepages_1G_request
```

### Example output

```
2
```

### Additional resources

- Allowing containers to consume Downward API objects

## 15.4. CONFIGURING HUGE PAGES AT BOOT TIME

Nodes must pre-allocate huge pages used in an OpenShift Container Platform cluster. There are two ways of reserving huge pages: at boot time and at run time. Reserving at boot time increases the possibility of success because the memory has not yet been significantly fragmented. The Node Tuning Operator currently supports boot time allocation of huge pages on specific nodes.

### Procedure

To minimize node reboots, the order of the steps below needs to be followed:

- Label all nodes that need the same huge pages setting by a label.

```
$ oc label node <node_using_hugepages> node-role.kubernetes.io/worker-hp=
```

- Create a file with the following content and name it **hugepages-tuned-boottime.yaml**:

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
 name: hugepages 1
 namespace: openshift-cluster-node-tuning-operator
spec:
 profile: 2
 - data:
 [main]
 summary=Boot time configuration for hugepages
 include=openshift-node
 [bootloader]
 cmdline.openshift_node_hugepages=hugepagesz=2M hugepages=50 3
```

```

name: openshift-node-hugepages

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

```

- ① Set the **name** of the Tuned resource to **hugepages**.
- ② Set the **profile** section to allocate huge pages.
- ③ Note the order of parameters is important as some platforms support huge pages of various sizes.
- ④ Enable machine config pool based matching.

### 3. Create the Tuned **hugepages** object

```
$ oc create -f hugepages-tuned-boottime.yaml
```

### 4. Create a file with the following content and name it **hugepages-mcp.yaml**:

```

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

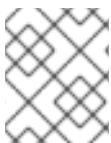
```

### 5. Create the machine config pool:

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

Given enough non-fragmented memory, all the nodes in the **worker-hp** machine config pool should now have 50 2Mi huge pages allocated.

```
$ oc get node <node_using_hugepages> -o jsonpath="{.status.allocatable.hugepages-2Mi}"
100Mi
```



#### NOTE

The TuneD bootloader plugin only supports Red Hat Enterprise Linux CoreOS (RHCOS) worker nodes.

## 15.5. DISABLING TRANSPARENT HUGE PAGES

Transparent Huge Pages (THP) attempt to automate most aspects of creating, managing, and using huge pages. Since THP automatically manages the huge pages, this is not always handled optimally for all types of workloads. THP can lead to performance regressions, since many applications handle huge pages on their own. Therefore, consider disabling THP. The following steps describe how to disable THP using the Node Tuning Operator (NTO).

### Procedure

1. Create a file with the following content and name it **thp-disable-tuned.yaml**:

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
 name: thp-workers-profile
 namespace: openshift-cluster-node-tuning-operator
spec:
 profile:
 - data: |
 [main]
 summary=Custom tuned profile for OpenShift to turn off THP on worker nodes
 include=openshift-node

 [vm]
 transparent_hugepages=never
 name: openshift-thp-never-worker

 recommend:
 - match:
 - label: node-role.kubernetes.io/worker
 priority: 25
 profile: openshift-thp-never-worker
```

2. Create the Tuned object:

```
$ oc create -f thp-disable-tuned.yaml
```

3. Check the list of active profiles:

```
$ oc get profile -n openshift-cluster-node-tuning-operator
```

### Verification

- Log in to one of the nodes and do a regular THP check to verify if the nodes applied the profile successfully:

```
$ cat /sys/kernel/mm/transparent_hugepage/enabled
```

### Example output

```
always madvise [never]
```

# CHAPTER 16. UNDERSTANDING LOW LATENCY TUNING FOR CLUSTER NODES

Edge computing has a key role in reducing latency and congestion problems and improving application performance for telco and 5G network applications. Maintaining a network architecture with the lowest possible latency is key for meeting the network performance requirements of 5G. Compared to 4G technology, with an average latency of 50 ms, 5G is targeted to reach latency of 1 ms or less. This reduction in latency boosts wireless throughput by a factor of 10.

## 16.1. ABOUT LOW LATENCY

Many of the deployed applications in the Telco space require low latency that can only tolerate zero packet loss. Tuning for zero packet loss helps mitigate the inherent issues that degrade network performance. For more information, see [Tuning for Zero Packet Loss in Red Hat OpenStack Platform \(RHOSP\)](#).

The Edge computing initiative also comes in to play for reducing latency rates. Think of it as being on the edge of the cloud and closer to the user. This greatly reduces the distance between the user and distant data centers, resulting in reduced application response times and performance latency.

Administrators must be able to manage their many Edge sites and local services in a centralized way so that all of the deployments can run at the lowest possible management cost. They also need an easy way to deploy and configure certain nodes of their cluster for real-time low latency and high-performance purposes. Low latency nodes are useful for applications such as Cloud-native Network Functions (CNF) and Data Plane Development Kit (DPDK).

OpenShift Container Platform currently provides mechanisms to tune software on an OpenShift Container Platform cluster for real-time running and low latency (around <20 microseconds reaction time). This includes tuning the kernel and OpenShift Container Platform set values, installing a kernel, and reconfiguring the machine. But this method requires setting up four different Operators and performing many configurations that, when done manually, is complex and could be prone to mistakes.

OpenShift Container Platform uses the Node Tuning Operator to implement automatic tuning to achieve low latency performance for OpenShift Container Platform applications. The cluster administrator uses this performance profile configuration that makes it easier to make these changes in a more reliable way. The administrator can specify whether to update the kernel to kernel-rt, reserve CPUs for cluster and operating system housekeeping duties, including pod infra containers, and isolate CPUs for application containers to run the workloads.

OpenShift Container Platform also supports workload hints for the Node Tuning Operator that can tune the **PerformanceProfile** to meet the demands of different industry environments. Workload hints are available for **highPowerConsumption** (very low latency at the cost of increased power consumption) and **realTime** (priority given to optimum latency). A combination of **true/false** settings for these hints can be used to deal with application-specific workload profiles and requirements.

Workload hints simplify the fine-tuning of performance to industry sector settings. Instead of a “one size fits all” approach, workload hints can cater to usage patterns such as placing priority on:

- Low latency
- Real-time capability
- Efficient use of power

Ideally, all of the previously listed items are prioritized. Some of these items come at the expense of

others however. The Node Tuning Operator is now aware of the workload expectations and better able to meet the demands of the workload. The cluster admin can now specify into which use case that workload falls. The Node Tuning Operator uses the **PerformanceProfile** to fine tune the performance settings for the workload.

The environment in which an application is operating influences its behavior. For a typical data center with no strict latency requirements, only minimal default tuning is needed that enables CPU partitioning for some high performance workload pods. For data centers and workloads where latency is a higher priority, measures are still taken to optimize power consumption. The most complicated cases are clusters close to latency-sensitive equipment such as manufacturing machinery and software-defined radios. This last class of deployment is often referred to as Far edge. For Far edge deployments, ultra-low latency is the ultimate priority, and is achieved at the expense of power management.

## 16.2. ABOUT HYPER-THREADING FOR LOW LATENCY AND REAL-TIME APPLICATIONS

Hyper-Threading is an Intel processor technology that allows a physical CPU processor core to function as two logical cores, executing two independent threads simultaneously. Hyper-Threading allows for better system throughput for certain workload types where parallel processing is beneficial. The default OpenShift Container Platform configuration expects Hyper-Threading to be enabled.

For telecommunications applications, it is important to design your application infrastructure to minimize latency as much as possible. Hyper-Threading can slow performance times and negatively affect throughput for compute-intensive workloads that require low latency. Disabling Hyper-Threading ensures predictable performance and can decrease processing times for these workloads.



### NOTE

Hyper-Threading implementation and configuration differs depending on the hardware you are running OpenShift Container Platform on. Consult the relevant host hardware tuning information for more details of the Hyper-Threading implementation specific to that hardware. Disabling Hyper-Threading can increase the cost per core of the cluster.

### Additional resources

- [Configuring Hyper-Threading for a cluster](#)

# CHAPTER 17. TUNING NODES FOR LOW LATENCY WITH THE PERFORMANCE PROFILE

Tune nodes for low latency by using the cluster performance profile. You can restrict CPUs for infra and application containers, configure huge pages, Hyper-Threading, and configure CPU partitions for latency-sensitive processes.

## 17.1. CREATING A PERFORMANCE PROFILE

You can create a cluster performance profile by using the Performance Profile Creator (PPC) tool. The PPC is a function of the Node Tuning Operator.

The PPC combines information about your cluster with user-supplied configurations to generate a performance profile that is appropriate to your hardware, topology and use-case.



### NOTE

Performance profiles are applicable only to bare-metal environments where the cluster has direct access to the underlying hardware resources. You can configure performances profiles for both single-node OpenShift and multi-node clusters.

The following is a high-level workflow for creating and applying a performance profile in your cluster:

- Create a machine config pool (MCP) for nodes that you want to target with performance configurations. In single-node OpenShift clusters, you must use the **master** MCP because there is only one node in the cluster.
- Gather information about your cluster using the **must-gather** command.
- Use the PPC tool to create a performance profile by using either of the following methods:
  - Run the PPC tool by using Podman.
  - Run the PPC tool by using a wrapper script.
- Configure the performance profile for your use case and apply the performance profile to your cluster.

### 17.1.1. About the Performance Profile Creator

The Performance Profile Creator (PPC) is a command-line tool, delivered with the Node Tuning Operator, that can help you to create a performance profile for your cluster.

Initially, you can use the PPC tool to process the **must-gather** data to display key performance configurations for your cluster, including the following information:

- NUMA cell partitioning with the allocated CPU IDs
- Hyper-Threading node configuration

You can use this information to help you configure the performance profile.

### Running the PPC

Specify performance configuration arguments to the PPC tool to generate a proposed performance profile that is appropriate for your hardware, topology, and use-case.

You can run the PPC by using one of the following methods:

- Run the PPC by using Podman
- Run the PPC by using the wrapper script



### NOTE

Using the wrapper script abstracts some of the more granular Podman tasks into an executable script. For example, the wrapper script handles tasks such as pulling and running the required container image, mounting directories into the container, and providing parameters directly to the container through Podman. Both methods achieve the same result.

## 17.1.2. Creating a machine config pool to target nodes for performance tuning

For multi-node clusters, you can define a machine config pool (MCP) to identify the target nodes that you want to configure with a performance profile.

In single-node OpenShift clusters, you must use the **master** MCP because there is only one node in the cluster. You do not need to create a separate MCP for single-node OpenShift clusters.

### Prerequisites

- You have **cluster-admin** role access.
- You installed the OpenShift CLI (**oc**).

### Procedure

1. Label the target nodes for configuration by running the following command:

```
$ oc label node <node_name> node-role.kubernetes.io/worker-cnf="" ①
```

- 1 Replace **<node\_name>** with the name of your node. This example applies the **worker-cnf** label.

2. Create a **MachineConfigPool** resource containing the target nodes:

- a. Create a YAML file that defines the **MachineConfigPool** resource:

#### Example mcp-worker-cnf.yaml file

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
 name: worker-cnf ①
 labels:
 machineconfiguration.openshift.io/role: worker-cnf ②
spec:
 machineConfigSelector:
```

```

matchExpressions:
- {
 key: machineconfiguration.openshift.io/role,
 operator: In,
 values: [worker, worker-cnf],
}
paused: false
nodeSelector:
matchLabels:
 node-role.kubernetes.io/worker-cnf: "" ③

```

- ① Specify a name for the **MachineConfigPool** resource.
- ② Specify a unique label for the machine config pool.
- ③ Specify the nodes with the target label that you defined.

b. Apply the **MachineConfigPool** resource by running the following command:

```
$ oc apply -f mcp-worker-cnf.yaml
```

#### Example output

```
machineconfigpool.machineconfiguration.openshift.io/worker-cnf created
```

### Verification

- Check the machine config pools in your cluster by running the following command:

```
$ oc get mcp
```

#### Example output

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT		
DEGRADED	MACHINECOUNT	AGE		
master	rendered-master-58433c7c3c1b4ed5ffef95234d451490	True		False
False	3	3	0	6h46m
worker	rendered-worker-168f52b168f151e4f853259729b6azc4	True		False
False	2	2	0	6h46m
worker-cnf	rendered-worker-cnf-168f52b168f151e4f853259729b6azc4	True		False
False	1	1	0	73s

### 17.1.3. Gathering data about your cluster for the PPC

The Performance Profile Creator (PPC) tool requires **must-gather** data. As a cluster administrator, run the **must-gather** command to capture information about your cluster.

#### Prerequisites

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

- You installed the OpenShift CLI (**oc**).
- You identified a target MCP that you want to configure with a performance profile.

## Procedure

1. Navigate to the directory where you want to store the **must-gather** data.
2. Collect cluster information by running the following command:

```
$ oc adm must-gather
```

The command creates a folder with the **must-gather** data in your local directory with a naming format similar to the following: **must-gather.local.1971646453781853027**.

3. Optional: Create a compressed file from the **must-gather** directory:

```
$ tar cvaf must-gather.tar.gz <must_gather_folder> ①
```

① Replace with the name of the **must-gather** data folder.



### NOTE

Compressed output is required if you are running the Performance Profile Creator wrapper script.

## Additional resources

- For more information about the **must-gather** tool, see [Gathering data about your cluster](#) .

### 17.1.4. Running the Performance Profile Creator using Podman

As a cluster administrator, you can use Podman with the Performance Profile Creator (PPC) to create a performance profile.

For more information about the PPC arguments, see the section "*Performance Profile Creator arguments*".



### IMPORTANT

The PPC uses the **must-gather** data from your cluster to create the performance profile. If you make any changes to your cluster, such as relabeling a node targeted for performance configuration, you must re-create the **must-gather** data before running PPC again.

## Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- A cluster installed on bare-metal hardware.
- You installed **podman** and the OpenShift CLI (**oc**).

- Access to the Node Tuning Operator image.
- You identified a machine config pool containing target nodes for configuration.
- You have access to the **must-gather** data for your cluster.

## Procedure

1. Check the machine config pool by running the following command:

```
$ oc get mcp
```

### Example output

	NAME	CONFIG	UPDATED	UPDATING	DEGRADED
	MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT		
	DEGRADED	MACHINECOUNT	AGE		
master	rendered-master-58433c8c3c0b4ed5feef95434d455490	3	0	True	False
False	3	3	8h		
worker	rendered-worker-668f56a164f151e4a853229729b6adc4	2	0	True	False
False	2	2	8h		
worker-cnf	rendered-worker-cnf-668f56a164f151e4a853229729b6adc4	1	0	True	False
False	1	1	79m		

2. Use Podman to authenticate to **registry.redhat.io** by running the following command:

```
$ podman login registry.redhat.io
```

```
Username: <user_name>
Password: <password>
```

3. Optional: Display help for the PPC tool by running the following command:

```
$ podman run --rm --entrypoint performance-profile-creator registry.redhat.io/openshift4/ose-cluster-node-tuning-rhel9-operator:v4.18 -h
```

### Example output

A tool that automates creation of Performance Profiles

Usage:

  performance-profile-creator [flags]

Flags:

--disable-ht	Disable Hyperthreading
-h, --help	help for performance-profile-creator
--info string	Show cluster information; requires --must-gather-dir-path, ignore the other arguments. [Valid values: log, json] (default "log")
--mcp-name string (required)	MCP name corresponding to the target machines
--must-gather-dir-path string	Must gather directory path (default "must-gather")
--offline-cpu-count int	Number of offline CPUs
--per-pod-power-management	Enable Per Pod Power Management

```
--power-consumption-mode string The power consumption mode. [Valid values:
default, low-latency, ultra-low-latency] (default "default")
--profile-name string Name of the performance profile to be created (default
"performance")
--reserved-cpu-count int Number of reserved CPUs (required)
--rt-kernel Enable Real Time Kernel (required)
--split-reserved-cpus-across-numa Split the Reserved CPUs across NUMA nodes
--topology-manager-policy string Kubelet Topology Manager Policy of the performance
profile to be created. [Valid values: single-numa-node, best-effort, restricted] (default
"restricted")
--user-level-networking Run with User level Networking(DPDK) enabled
```

4. To display information about the cluster, run the PPC tool with the **log** argument by running the following command:

```
$ podman run --entrypoint performance-profile-creator -v <path_to_must_gather>:/must-gather:z registry.redhat.io/openshift4/ose-cluster-node-tuning-rhel9-operator:v4.18 --info log
--must-gather-dir-path /must-gather
```

- **--entrypoint performance-profile-creator** defines the performance profile creator as a new entry point to **podman**.
- **-v <path\_to\_must\_gather>** specifies the path to either of the following components:
  - The directory containing the **must-gather** data.
  - An existing directory containing the **must-gather** decompressed .tar file.
- **--info log** specifies a value for the output format.

### Example output

```
level=info msg="Cluster info:"
level=info msg="MCP 'master' nodes:"
level=info msg="---"
level=info msg="MCP 'worker' nodes:"
level=info msg="Node: host.example.com (NUMA cells: 1, HT: true)"
level=info msg="NUMA cell 0 : [0 1 2 3]"
level=info msg="CPU(s): 4"
level=info msg="Node: host1.example.com (NUMA cells: 1, HT: true)"
level=info msg="NUMA cell 0 : [0 1 2 3]"
level=info msg="CPU(s): 4"
level=info msg="---"
level=info msg="MCP 'worker-cnf' nodes:"
level=info msg="Node: host2.example.com (NUMA cells: 1, HT: true)"
level=info msg="NUMA cell 0 : [0 1 2 3]"
level=info msg="CPU(s): 4"
level=info msg="---"
```

5. Create a performance profile by running the following command. The example uses sample PPC arguments and values:

```
$ podman run --entrypoint performance-profile-creator -v <path_to_must_gather>:/must-gather:z registry.redhat.io/openshift4/ose-cluster-node-tuning-rhel9-operator:v4.18 --mcp-name=worker-cnf --reserved-cpu-count=1 --rt-kernel=true --split-reserved-cpus-across-
```

```
numa=false --must-gather-dir-path /must-gather --power-consumption-mode=ultra-low-latency --offlined-cpu-count=1 > my-performance-profile.yaml
```

- **-v <path\_to\_must\_gather>** specifies the path to either of the following components:
  - The directory containing the **must-gather** data.
  - The directory containing the **must-gather** decompressed .tar file.
- **--mcp-name=worker-cnf** specifies the **worker-cnf** machine config pool.
- **--reserved-cpu-count=1** specifies one reserved CPU.
- **--rt-kernel=true** enables the real-time kernel.
- **--split-reserved-cpus-across-numa=false** disables reserved CPUs splitting across NUMA nodes.
- **--power-consumption-mode=ultra-low-latency** specifies minimal latency at the cost of increased power consumption.
- **--offlined-cpu-count=1** specifies one offlined CPU.



#### NOTE

The **mcp-name** argument in this example is set to **worker-cnf** based on the output of the command **oc get mcp**. For single-node OpenShift use **--mcp-name=master**.

#### Example output

```
level=info msg="Nodes targeted by worker-cnf MCP are: [worker-2]"
level=info msg="NUMA cell(s): 1"
level=info msg="NUMA cell 0 : [0 1 2 3]"
level=info msg="CPU(s): 4"
level=info msg="1 reserved CPUs allocated: 0 "
level=info msg="2 isolated CPUs allocated: 2-3"
level=info msg="Additional Kernel Args based on configuration: []"
```

6. Review the created YAML file by running the following command:

```
$ cat my-performance-profile.yaml
```

#### Example output

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: performance
spec:
 cpu:
 isolated: 2-3
 offlined: "1"
 reserved: "0"
```

```

machineConfigPoolSelector:
 machineconfiguration.openshift.io/role: worker-cnf
nodeSelector:
 node-role.kubernetes.io/worker-cnf: ""
numa:
 topologyPolicy: restricted
realTimeKernel:
 enabled: true
workloadHints:
 highPowerConsumption: true
 perPodPowerManagement: false
 realTime: true

```

7. Apply the generated profile:

```
$ oc apply -f my-performance-profile.yaml
```

#### Example output

```
performanceprofile.performance.openshift.io/performance created
```

### 17.1.5. Running the Performance Profile Creator wrapper script

The wrapper script simplifies the process of creating a performance profile with the Performance Profile Creator (PPC) tool. The script handles tasks such as pulling and running the required container image, mounting directories into the container, and providing parameters directly to the container through Podman.

For more information about the Performance Profile Creator arguments, see the section “*Performance Profile Creator arguments*”.



#### IMPORTANT

The PPC uses the **must-gather** data from your cluster to create the performance profile. If you make any changes to your cluster, such as relabeling a node targeted for performance configuration, you must re-create the **must-gather** data before running PPC again.

#### Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- A cluster installed on bare-metal hardware.
- You installed **podman** and the OpenShift CLI (**oc**).
- Access to the Node Tuning Operator image.
- You identified a machine config pool containing target nodes for configuration.
- Access to the **must-gather** tarball.

#### Procedure

1. Create a file on your local machine named, for example, **run-perf-profile-creator.sh**:

```
$ vi run-perf-profile-creator.sh
```

2. Paste the following code into the file:

```
#!/bin/bash

readonly CONTAINER_RUNTIME=${CONTAINER_RUNTIME:-podman}
readonly CURRENT_SCRIPT=$(basename "$0")
readonly CMD="${CONTAINER_RUNTIME} run --entrypoint performance-profile-creator"
readonly IMG_EXISTS_CMD="${CONTAINER_RUNTIME} image exists"
readonly IMG_PULL_CMD="${CONTAINER_RUNTIME} image pull"
readonly MUST_GATHER_VOL="/must-gather"

NTO_IMG="registry.redhat.io/openshift4/ose-cluster-node-tuning-rhel9-operator:v4.18"
MG_TARBALL=""
DATA_DIR=""

usage() {
 print "Wrapper usage:"
 print " ${CURRENT_SCRIPT} [-h] [-p image][-t path] -- [performance-profile-creator flags]"
 print ""
 print "Options:"
 print " -h help for ${CURRENT_SCRIPT}"
 print " -p Node Tuning Operator image"
 print " -t path to a must-gather tarball"

 ${IMG_EXISTS_CMD} "${NTO_IMG}" && ${CMD} "${NTO_IMG}" -h
}

function cleanup {
 [-d "${DATA_DIR}"] && rm -rf "${DATA_DIR}"
}
trap cleanup EXIT

exit_error() {
 print "error: $*"
 usage
 exit 1
}

print() {
 echo "$*" >&2
}

check_requirements() {
 ${IMG_EXISTS_CMD} "${NTO_IMG}" || ${IMG_PULL_CMD} "${NTO_IMG}" || \
 exit_error "Node Tuning Operator image not found"

 [-n "${MG_TARBALL}"] || exit_error "Must-gather tarball file path is mandatory"
 [-f "${MG_TARBALL}"] || exit_error "Must-gather tarball file not found"

 DATA_DIR=$(mktemp -d -t "${CURRENT_SCRIPT}XXXX") || exit_error "Cannot create the
data directory"
 tar -zxf "${MG_TARBALL}" --directory "${DATA_DIR}" || exit_error "Cannot decompress the
```

```

must-gather tarball"
chmod a+rx "${DATA_DIR}"

return 0
}

main() {
while getopts ':hp:t:' OPT; do
case "${OPT}" in
h)
usage
exit 0
;;
p)
NTO_IMG="${OPTARG}"
;;
t)
MG_TARBALL="${OPTARG}"
;;
?)
exit_error "invalid argument: ${OPTARG}"
;;
esac
done
shift $((OPTIND - 1))

check_requirements || exit 1

${CMD} -v "${DATA_DIR}:${MUST_GATHER_VOL}:z" "${NTO_IMG}" "$@" --must-gather-
dir-path "${MUST_GATHER_VOL}"
echo "" 1>&2
}

main "$@"

```

- Add execute permissions for everyone on this script:

```
$ chmod a+x run-perf-profile-creator.sh
```

- Use Podman to authenticate to **registry.redhat.io** by running the following command:

```
$ podman login registry.redhat.io
```

```
Username: <user_name>
Password: <password>
```

- Optional: Display help for the PPC tool by running the following command:

```
$./run-perf-profile-creator.sh -h
```

### Example output

```
Wrapper usage:
run-perf-profile-creator.sh [-h] [-p image][-t path] -- [performance-profile-creator flags]
```

**Options:**

-h	help for run-perf-profile-creator.sh
-p	Node Tuning Operator image
-t	path to a must-gather tarball

A tool that automates creation of Performance Profiles

**Usage:**

```
performance-profile-creator [flags]
```

**Flags:**

--disable-ht	Disable Hyperthreading
-h, --help	help for performance-profile-creator
--info string	Show cluster information; requires --must-gather-dir-path, ignore the other arguments. [Valid values: log, json] (default "log")
--mcp-name string	MCP name corresponding to the target machines (required)
--must-gather-dir-path string	Must gather directory path (default "must-gather")
--offlined-cpu-count int	Number of offlined CPUs
--per-pod-power-management	Enable Per Pod Power Management
--power-consumption-mode string	The power consumption mode. [Valid values: default, low-latency, ultra-low-latency] (default "default")
--profile-name string	Name of the performance profile to be created (default "performance")
--reserved-cpu-count int	Number of reserved CPUs (required)
--rt-kernel	Enable Real Time Kernel (required)
--split-reserved-cpus-across-numa	Split the Reserved CPUs across NUMA nodes
--topology-manager-policy string	Kubelet Topology Manager Policy of the performance profile to be created. [Valid values: single-numa-node, best-effort, restricted] (default "restricted")
--user-level-networking	Run with User level Networking(DPDK) enabled
--enable-hardware-tuning	Enable setting maximum CPU frequencies

**NOTE**

You can optionally set a path for the Node Tuning Operator image using the **-p** option. If you do not set a path, the wrapper script uses the default image: **registry.redhat.io/openshift4/ose-cluster-node-tuning-rhel9-operator:v4.18**.

- To display information about the cluster, run the PPC tool with the **log** argument by running the following command:

```
$./run-perf-profile-creator.sh -t /<path_to_must_gather_dir>/must-gather.tar.gz -- --info=log
```

- t /<path\_to\_must\_gather\_dir>/must-gather.tar.gz** specifies the path to directory containing the must-gather tarball. This is a required argument for the wrapper script.

**Example output**

```
level=info msg="Cluster info:"
level=info msg="MCP 'master' nodes:"
level=info msg=---
level=info msg="MCP 'worker' nodes:"
level=info msg="Node: host.example.com (NUMA cells: 1, HT: true)"
```

```

level=info msg="NUMA cell 0 : [0 1 2 3]"
level=info msg="CPU(s): 4"
level=info msg="Node: host1.example.com (NUMA cells: 1, HT: true)"
level=info msg="NUMA cell 0 : [0 1 2 3]"
level=info msg="CPU(s): 4"
level=info msg=---
level=info msg="MCP 'worker-cnf' nodes:"
level=info msg="Node: host2.example.com (NUMA cells: 1, HT: true)"
level=info msg="NUMA cell 0 : [0 1 2 3]"
level=info msg="CPU(s): 4"
level=info msg=---

```

7. Create a performance profile by running the following command.

```
$./run-perf-profile-creator.sh -t /path-to-must-gather/must-gather.tar.gz -- --mcp-name=worker-cnf --reserved-cpu-count=1 --rt-kernel=true --split-reserved-cpus-across-numa=false --power-consumption-mode=ultra-low-latency --offlined-cpu-count=1 > my-performance-profile.yaml
```

This example uses sample PPC arguments and values.

- **--mcp-name=worker-cnf** specifies the **worker-cnf** machine config pool.
- **--reserved-cpu-count=1** specifies one reserved CPU.
- **--rt-kernel=true** enables the real-time kernel.
- **--split-reserved-cpus-across-numa=false** disables reserved CPUs splitting across NUMA nodes.
- **--power-consumption-mode=ultra-low-latency** specifies minimal latency at the cost of increased power consumption.
- **--offlined-cpu-count=1** specifies one offlined CPUs.



#### NOTE

The **mcp-name** argument in this example is set to **worker-cnf** based on the output of the command **oc get mcp**. For single-node OpenShift use **--mcp-name=master**.

8. Review the created YAML file by running the following command:

```
$ cat my-performance-profile.yaml
```

#### Example output

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: performance
spec:
 cpu:

```

```

isolated: 2-3
offlined: "1"
reserved: "0"
machineConfigPoolSelector:
 machineconfiguration.openshift.io/role: worker-cnf
nodeSelector:
 node-role.kubernetes.io/worker-cnf: ""
numa:
 topologyPolicy: restricted
realTimeKernel:
 enabled: true
workloadHints:
 highPowerConsumption: true
 perPodPowerManagement: false
 realTime: true

```

9. Apply the generated profile:

```
$ oc apply -f my-performance-profile.yaml
```

#### Example output

```
performanceprofile.performance.openshift.io/performance created
```

### 17.1.6. Performance Profile Creator arguments

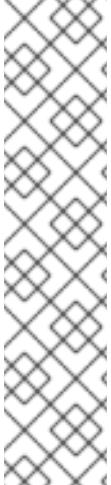
Table 17.1. Required Performance Profile Creator arguments

Argument	Description
<b>mcp-name</b>	Name for MCP; for example, <b>worker-cnf</b> corresponding to the target machines.
<b>must-gather-dir-path</b>	The path of the must gather directory.  This argument is only required if you run the PPC tool by using Podman. If you use the PPC with the wrapper script, do not use this argument. Instead, specify the directory path to the <b>must-gather</b> tarball by using the <b>-t</b> option for the wrapper script.
<b>reserved-cpu-count</b>	Number of reserved CPUs. Use a natural number greater than zero.
<b>rt-kernel</b>	Enables real-time kernel.  Possible values: <b>true</b> or <b>false</b> .

Table 17.2. Optional Performance Profile Creator arguments

Argument	Description
----------	-------------

Argument	Description
<b>disable-ht</b>	<p>Disable Hyper-Threading.</p> <p>Possible values: <b>true</b> or <b>false</b>.</p> <p>Default: <b>false</b>.</p> <div style="background-color: #ffffcc; padding: 10px;">  <p><b>WARNING</b></p> <p>If this argument is set to <b>true</b> you should not disable Hyper-Threading in the BIOS. Disabling Hyper-Threading is accomplished with a kernel command-line argument.</p> </div>
<b>enable-hardware-tuning</b>	<p>Enable the setting of maximum CPU frequencies.</p> <p>To enable this feature, set the maximum frequency for applications running on isolated and reserved CPUs for both of the following fields:</p> <ul style="list-style-type: none"> <li>• <b>spec.hardwareTuning.isolatedCpuFreq</b></li> <li>• <b>spec.hardwareTuning.reservedCpuFreq</b></li> </ul> <p>This is an advanced feature. If you configure hardware tuning, the generated <b>PerformanceProfile</b> includes warnings and guidance on how to set frequency settings.</p>
<b>info</b>	<p>This captures cluster information. This argument also requires the <b>must-gather-dir-path</b> argument. If any other arguments are set they are ignored.</p> <p>Possible values:</p> <ul style="list-style-type: none"> <li>• <b>log</b></li> <li>• <b>JSON</b></li> </ul> <p>Default: <b>log</b>.</p>

Argument	Description
<b>offlined-cpu-count</b>	<p>Number of offlined CPUs.</p> <p> <b>NOTE</b></p> <p>Use a natural number greater than zero. If not enough logical processors are offlined, then error messages are logged. The messages are:</p> <ul style="list-style-type: none"> <li>─ Error: failed to compute the reserved and isolated CPUs: please ensure that reserved-cpu-count plus offlined-cpu-count should be in the range [0,1]</li> <li>─ Error: failed to compute the reserved and isolated CPUs: please specify the offlined CPU count in the range [0,1]</li> </ul>
<b>power-consumption-mode</b>	<p>The power consumption mode.</p> <p>Possible values:</p> <ul style="list-style-type: none"> <li>● <b>default</b>: Performance achieved through CPU partitioning only.</li> <li>● <b>low-latency</b>: Enhanced measures to improve latency.</li> <li>● <b>ultra-low-latency</b>: Priority given to optimal latency, at the expense of power management.</li> </ul> <p>Default: <b>default</b>.</p>
<b>per-pod-power-management</b>	<p>Enable per pod power management. You cannot use this argument if you configured <b>ultra-low-latency</b> as the power consumption mode.</p> <p>Possible values: <b>true</b> or <b>false</b>.</p> <p>Default: <b>false</b>.</p>
<b>profile-name</b>	<p>Name of the performance profile to create.</p> <p>Default: <b>performance</b>.</p>
<b>split-reserved-cpus-across-numa</b>	<p>Split the reserved CPUs across NUMA nodes.</p> <p>Possible values: <b>true</b> or <b>false</b>.</p> <p>Default: <b>false</b>.</p>

Argument	Description
<b>topology-manager-policy</b>	Kubelet Topology Manager policy of the performance profile to be created.  Possible values: <ul style="list-style-type: none"><li>• <b>single-numa-node</b></li><li>• <b>best-effort</b></li><li>• <b>restricted</b></li></ul> Default: <b>restricted</b> .
<b>user-level-networking</b>	Run with user level networking (DPDK) enabled.  Possible values: <b>true</b> or <b>false</b> .  Default: <b>false</b> .

## 17.1.7. Reference performance profiles

Use the following reference performance profiles as the basis to develop your own custom profiles.

### 17.1.7.1. Performance profile template for clusters that use OVS-DPDK on OpenStack

To maximize machine performance in a cluster that uses Open vSwitch with the Data Plane Development Kit (OVS-DPDK) on Red Hat OpenStack Platform (RHOSP), you can use a performance profile.

You can use the following performance profile template to create a profile for your deployment.

#### Performance profile template for clusters that use OVS-DPDK

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: cnf-performanceprofile
spec:
 additionalKernelArgs:
 - nmi_watchdog=0
 - audit=0
 - mce=off
 - processor.max_cstate=1
 - idle=poll
 - intel_idle.max_cstate=0
 - default_hugepagesz=1GB
 - hugepagesz=1G
 - intel_iommu=on
 cpu:
 isolated: <CPU_ISOLATED>
 reserved: <CPU_RESERVED>
 hugepages:
 defaultHugepagesSize: 1G
 pages:
```

```

- count: <HUGEPAGES_COUNT>
 node: 0
 size: 1G
nodeSelector:
 node-role.kubernetes.io/worker: ""
globallyDisableIRQLoadBalancing: true
realTimeKernel:
 enabled: false

```

Insert values that are appropriate for your configuration for the **CPU\_ISOLATED**, **CPU\_RESERVED**, and **HUGEPAGES\_COUNT** keys.

### 17.1.7.2. Telco RAN DU reference design performance profile

The following performance profile configures node-level performance settings for OpenShift Container Platform clusters on commodity hardware to host telco RAN DU workloads.

#### Telco RAN DU reference design performance profile

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 # if you change this name make sure the 'include' line in TunedPerformancePatch.yaml
 # matches this name: include=openshift-node-performance-${PerformanceProfile.metadata.name}
 # Also in file 'validatorCRs/informDuValidator.yaml':
 # name: 50-performance-${PerformanceProfile.metadata.name}
 name: openshift-node-performance-profile
 annotations:
 ran.openshift.io/reference-configuration: "ran-du.redhat.com"
spec:
 additionalKernelArgs:
 - "rcupdate.rcu_normal_after_boot=0"
 - "efi=runtime"
 - "vfio_pci.enable_sriov=1"
 - "vfio_pci.disable_idle_d3=1"
 - "module_blacklist=irdma"
 cpu:
 isolated: $isolated
 reserved: $reserved
 hugepages:
 defaultHugepagesSize: $defaultHugepagesSize
 pages:
 - size: $size
 count: $count
 node: $node
 machineConfigPoolSelector:
 pools.operator.machineconfiguration.openshift.io/$mcp: ""
 nodeSelector:
 node-role.kubernetes.io/$mcp: ""
 numa:
 topologyPolicy: "restricted"
 # To use the standard (non-realtime) kernel, set enabled to false
 realTimeKernel:
 enabled: true
 workloadHints:

```

```
WorkloadHints defines the set of upper level flags for different type of workloads.
See https://github.com/openshift/cluster-node-tuning-
operator/blob/master/docs/performanceprofile/performance_profile.md#workloadhints
for detailed descriptions of each item.
The configuration below is set for a low latency, performance mode.
realTime: true
highPowerConsumption: false
perPodPowerManagement: false
```

### 17.1.7.3. Telco core reference design performance profile

The following performance profile configures node-level performance settings for OpenShift Container Platform clusters on commodity hardware to host telco core workloads.

#### Telco core reference design performance profile

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 # if you change this name make sure the 'include' line in TunedPerformancePatch.yaml
 # matches this name: include=openshift-node-performance-${PerformanceProfile.metadata.name}
 # Also in file 'validatorCRs/informDuValidator.yaml':
 # name: 50-performance-${PerformanceProfile.metadata.name}
 name: openshift-node-performance-profile
 annotations:
 ran.openshift.io/reference-configuration: "ran-du.redhat.com"
spec:
 additionalKernelArgs:
 - "rcupdate.rcu_normal_after_boot=0"
 - "efi=runtime"
 - "vfio_pci.enable_sriov=1"
 - "vfio_pci.disable_idle_d3=1"
 - "module_blacklist=irdma"
 cpu:
 isolated: $isolated
 reserved: $reserved
 hugepages:
 defaultHugepagesSize: $defaultHugepagesSize
 pages:
 - size: $size
 count: $count
 node: $node
 machineConfigPoolSelector:
 pools.operator.machineconfiguration.openshift.io/$mcp: ""
 nodeSelector:
 node-role.kubernetes.io/$mcp: ""
 numa:
 topologyPolicy: "restricted"
 # To use the standard (non-realtime) kernel, set enabled to false
 realTimeKernel:
 enabled: true
 workloadHints:
 # WorkloadHints defines the set of upper level flags for different type of workloads.
 # See https://github.com/openshift/cluster-node-tuning-
operator/blob/master/docs/performanceprofile/performance_profile.md#workloadhints
```

```
for detailed descriptions of each item.
The configuration below is set for a low latency, performance mode.
realTime: true
highPowerConsumption: false
perPodPowerManagement: false
```

## 17.2. SUPPORTED PERFORMANCE PROFILE API VERSIONS

The Node Tuning Operator supports **v2**, **v1**, and **v1alpha1** for the performance profile **apiVersion** field. The v1 and v1alpha1 APIs are identical. The v2 API includes an optional boolean field **globallyDisableIRQLoadBalancing** with a default value of **false**.

### Upgrading the performance profile to use device interrupt processing

When you upgrade the Node Tuning Operator performance profile custom resource definition (CRD) from v1 or v1alpha1 to v2, **globallyDisableIRQLoadBalancing** is set to **true** on existing profiles.



#### NOTE

**globallyDisableIRQLoadBalancing** toggles whether IRQ load balancing will be disabled for the Isolated CPU set. When the option is set to **true** it disables IRQ load balancing for the Isolated CPU set. Setting the option to **false** allows the IRQs to be balanced across all CPUs.

### Upgrading Node Tuning Operator API from v1alpha1 to v1

When upgrading Node Tuning Operator API version from v1alpha1 to v1, the v1alpha1 performance profiles are converted on-the-fly using a "None" Conversion strategy and served to the Node Tuning Operator with API version v1.

### Upgrading Node Tuning Operator API from v1alpha1 or v1 to v2

When upgrading from an older Node Tuning Operator API version, the existing v1 and v1alpha1 performance profiles are converted using a conversion webhook that injects the **globallyDisableIRQLoadBalancing** field with a value of **true**.

## 17.3. CONFIGURING NODE POWER CONSUMPTION AND REALTIME PROCESSING WITH WORKLOAD HINTS

### Procedure

- Create a **PerformanceProfile** appropriate for the environment's hardware and topology by using the Performance Profile Creator (PPC) tool. The following table describes the possible values set for the **power-consumption-mode** flag associated with the PPC tool and the workload hint that is applied.

Table 17.3. Impact of combinations of power consumption and real-time settings on latency

Performance Profile creator setting	Hint	Environment	Description
-------------------------------------	------	-------------	-------------

Performance Profile creator setting	Hint	Environment	Description
Default	workloadHints: highPowerConsumption: false realTime: false	High throughput cluster without latency requirements	Performance achieved through CPU partitioning only.
Low-latency	workloadHints: highPowerConsumption: false realTime: true	Regional data-centers	Both energy savings and low-latency are desirable: compromise between power management, latency and throughput.
Ultra-low-latency	workloadHints: highPowerConsumption: true realTime: true	Far edge clusters, latency critical workloads	Optimized for absolute minimal latency and maximum determinism at the cost of increased power consumption.
Per-pod power management	workloadHints: realTime: true highPowerConsumption: false perPodPowerManagement: true	Critical and non-critical workloads	Allows for power management per pod.

## Example

The following configuration is commonly used in a telco RAN DU deployment.

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: workload-hints
spec:
 ...
 workloadHints:
 realTime: true
 highPowerConsumption: false
 perPodPowerManagement: false ①
```

① Disables some debugging and monitoring features that can affect system latency.

**NOTE**

When the **realTime** workload hint flag is set to **true** in a performance profile, add the **cpu-quota.rio.io: disable** annotation to every guaranteed pod with pinned CPUs. This annotation is necessary to prevent the degradation of the process performance within the pod. If the **realTime** workload hint is not explicitly set, it defaults to **true**.

For more information how combinations of power consumption and real-time settings impact latency, see [Understanding workload hints](#).

## 17.4. CONFIGURING POWER SAVING FOR NODES THAT RUN COLOCATED HIGH AND LOW PRIORITY WORKLOADS

You can enable power savings for a node that has low priority workloads that are colocated with high priority workloads without impacting the latency or throughput of the high priority workloads. Power saving is possible without modifications to the workloads themselves.

**IMPORTANT**

The feature is supported on Intel Ice Lake and later generations of Intel CPUs. The capabilities of the processor might impact the latency and throughput of the high priority workloads.

### Prerequisites

- You enabled C-states and operating system controlled P-states in the BIOS

### Procedure

- 1 Generate a **PerformanceProfile** with the **per-pod-power-management** argument set to **true**:

```
$ podman run --entrypoint performance-profile-creator -v \
/must-gather:/must-gather:z registry.redhat.io/openshift4/ose-cluster-node-tuning-rhel9-
operator:v4.18 \
--mcp-name=worker-cnf --reserved-cpu-count=20 --rt-kernel=true \
--split-reserved-cpus-across-numa=false --topology-manager-policy=single-numa-node \
--must-gather-dir-path /must-gather --power-consumption-mode=low-latency \ ①
--per-pod-power-management=true > my-performance-profile.yaml
```

- 1 The **power-consumption-mode** argument must be **default** or **low-latency** when the **per-pod-power-management** argument is set to **true**.

### Example PerformanceProfile with perPodPowerManagement

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: performance
spec:
 [....]
workloadHints:
```

```
realTime: true
highPowerConsumption: false
perPodPowerManagement: true
```

- Set the default **cpufreq** governor as an additional kernel argument in the **PerformanceProfile** custom resource (CR):

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: performance
spec:
 ...
 additionalKernelArgs:
 - cpufreq.default_governor=schedutil ①
```

- ① Using the **schedutil** governor is recommended, however, you can use other governors such as the **ondemand** or **powersave** governors.

- Set the maximum CPU frequency in the **TunedPerformancePatch** CR:

```
spec:
profile:
- data: |
 [sysfs]
 /sys/devices/system/cpu/intel_pstate/max_perf_pct = <x> ①
```

- ① The **max\_perf\_pct** controls the maximum frequency that the **cpufreq** driver is allowed to set as a percentage of the maximum supported cpu frequency. This value applies to all CPUs. You can check the maximum supported frequency in **/sys/devices/system/cpu/cpu0/cpufreq/cpuinfo\_max\_freq**. As a starting point, you can use a percentage that caps all CPUs at the **All Cores Turbo** frequency. The **All Cores Turbo** frequency is the frequency that all cores will run at when the cores are all fully occupied.

## Additional resources

- [About the Performance Profile Creator](#)
- [Disabling power saving mode for high priority pods](#)
- [Managing device interrupt processing for guaranteed pod isolated CPUs](#)

## 17.5. RESTRICTING CPUS FOR INFRA AND APPLICATION CONTAINERS

Generic housekeeping and workload tasks use CPUs in a way that may impact latency-sensitive processes. By default, the container runtime uses all online CPUs to run all containers together, which can result in context switches and spikes in latency. Partitioning the CPUs prevents noisy processes from interfering with latency-sensitive processes by separating them from each other. The following table describes how processes run on a CPU after you have tuned the node using the Node Tuning Operator:

Table 17.4. Process' CPU assignments

Process type	Details
<b>Burstable</b> and <b>BestEffort</b> pods	Runs on any CPU except where low latency workload is running
Infrastructure pods	Runs on any CPU except where low latency workload is running
Interrupts	Redirects to reserved CPUs (optional in OpenShift Container Platform 4.7 and later)
Kernel processes	Pins to reserved CPUs
Latency-sensitive workload pods	Pins to a specific set of exclusive CPUs from the isolated pool
OS processes/systemd services	Pins to reserved CPUs

The allocatable capacity of cores on a node for pods of all QoS process types, **Burstable**, **BestEffort**, or **Guaranteed**, is equal to the capacity of the isolated pool. The capacity of the reserved pool is removed from the node's total core capacity for use by the cluster and operating system housekeeping duties.

### Example 1

A node features a capacity of 100 cores. Using a performance profile, the cluster administrator allocates 50 cores to the isolated pool and 50 cores to the reserved pool. The cluster administrator assigns 25 cores to QoS **Guaranteed** pods and 25 cores for **BestEffort** or **Burstable** pods. This matches the capacity of the isolated pool.

### Example 2

A node features a capacity of 100 cores. Using a performance profile, the cluster administrator allocates 50 cores to the isolated pool and 50 cores to the reserved pool. The cluster administrator assigns 50 cores to QoS **Guaranteed** pods and one core for **BestEffort** or **Burstable** pods. This exceeds the capacity of the isolated pool by one core. Pod scheduling fails because of insufficient CPU capacity.

The exact partitioning pattern to use depends on many factors like hardware, workload characteristics and the expected system load. Some sample use cases are as follows:

- If the latency-sensitive workload uses specific hardware, such as a network interface controller (NIC), ensure that the CPUs in the isolated pool are as close as possible to this hardware. At a minimum, you should place the workload in the same Non-Uniform Memory Access (NUMA) node.
- The reserved pool is used for handling all interrupts. When depending on system networking, allocate a sufficiently-sized reserve pool to handle all the incoming packet interrupts. In 4.18 and later versions, workloads can optionally be labeled as sensitive.

The decision regarding which specific CPUs should be used for reserved and isolated partitions requires detailed analysis and measurements. Factors like NUMA affinity of devices and memory play a role. The selection also depends on the workload architecture and the specific use case.



## IMPORTANT

The reserved and isolated CPU pools must not overlap and together must span all available cores in the worker node.

To ensure that housekeeping tasks and workloads do not interfere with each other, specify two groups of CPUs in the **spec** section of the performance profile.

- **isolated** - Specifies the CPUs for the application container workloads. These CPUs have the lowest latency. Processes in this group have no interruptions and can, for example, reach much higher DPDK zero packet loss bandwidth.
- **reserved** - Specifies the CPUs for the cluster and operating system housekeeping duties. Threads in the **reserved** group are often busy. Do not run latency-sensitive applications in the **reserved** group. Latency-sensitive applications run in the **isolated** group.

## Procedure

1. Create a performance profile appropriate for the environment's hardware and topology.
2. Add the **reserved** and **isolated** parameters with the CPUs you want reserved and isolated for the infra and application containers:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: infra-cpus
spec:
 cpu:
 reserved: "0-4,9" ①
 isolated: "5-8" ②
 nodeSelector: ③
 node-role.kubernetes.io/worker: ""
```

- ① Specify which CPUs are for infra containers to perform cluster and operating system housekeeping duties.
- ② Specify which CPUs are for application containers to run workloads.
- ③ Optional: Specify a node selector to apply the performance profile to specific nodes.

## 17.6. CONFIGURING HYPER-THREADING FOR A CLUSTER

To configure Hyper-Threading for an OpenShift Container Platform cluster, set the CPU threads in the performance profile to the same cores that are configured for the reserved or isolated CPU pools.



## NOTE

If you configure a performance profile, and subsequently change the Hyper-Threading configuration for the host, ensure that you update the CPU **isolated** and **reserved** fields in the **PerformanceProfile** YAML to match the new configuration.



## WARNING

Disabling a previously enabled host Hyper-Threading configuration can cause the CPU core IDs listed in the **PerformanceProfile** YAML to be incorrect. This incorrect configuration can cause the node to become unavailable because the listed CPUs can no longer be found.

## Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- Install the OpenShift CLI (oc).

## Procedure

1. Ascertain which threads are running on what CPUs for the host you want to configure.  
You can view which threads are running on the host CPUs by logging in to the cluster and running the following command:

```
$ lscpu --all --extended
```

### Example output

CPU	NODE	SOCKET	CORE	L1d:L1i:L2:L3	ONLINE	MAXMHZ	MINMHZ
0	0	0	0	0:0:0:0	yes	4800.0000	400.0000
1	0	0	1	1:1:1:0	yes	4800.0000	400.0000
2	0	0	2	2:2:2:0	yes	4800.0000	400.0000
3	0	0	3	3:3:3:0	yes	4800.0000	400.0000
4	0	0	0	0:0:0:0	yes	4800.0000	400.0000
5	0	0	1	1:1:1:0	yes	4800.0000	400.0000
6	0	0	2	2:2:2:0	yes	4800.0000	400.0000
7	0	0	3	3:3:3:0	yes	4800.0000	400.0000

In this example, there are eight logical CPU cores running on four physical CPU cores. CPU0 and CPU4 are running on physical Core0, CPU1 and CPU5 are running on physical Core 1, and so on.

Alternatively, to view the threads that are set for a particular physical CPU core (**cpu0** in the example below), open a shell prompt and run the following:

```
$ cat /sys/devices/system/cpu/cpu0/topology/thread_siblings_list
```

### Example output

```
0-4
```

2. Apply the isolated and reserved CPUs in the **PerformanceProfile** YAML. For example, you can set logical cores CPU0 and CPU4 as **isolated**, and logical cores CPU1 to CPU3 and CPU5 to CPU7 as **reserved**. When you configure reserved and isolated CPUs, the infra containers in

pods use the reserved CPUs and the application containers use the isolated CPUs.

```
...
cpu:
isolated: 0,4
reserved: 1-3,5-7
...
```



### NOTE

The reserved and isolated CPU pools must not overlap and together must span all available cores in the worker node.



### IMPORTANT

Hyper-Threading is enabled by default on most Intel processors. If you enable Hyper-Threading, all threads processed by a particular core must be isolated or processed on the same core.

When Hyper-Threading is enabled, all guaranteed pods must use multiples of the simultaneous multi-threading (SMT) level to avoid a "noisy neighbor" situation that can cause the pod to fail. See [Static policy options](#) for more information.

#### 17.6.1. Disabling Hyper-Threading for low latency applications

When configuring clusters for low latency processing, consider whether you want to disable Hyper-Threading before you deploy the cluster. To disable Hyper-Threading, perform the following steps:

1. Create a performance profile that is appropriate for your hardware and topology.
2. Set **nosmt** as an additional kernel argument. The following example performance profile illustrates this setting:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: example-performanceprofile
spec:
 additionalKernelArgs:
 - nmi_watchdog=0
 - audit=0
 - mce=off
 - processor.max_cstate=1
 - idle=poll
 - intel_idle.max_cstate=0
 - nosmt
 cpu:
 isolated: 2-3
 reserved: 0-1
 hugepages:
 defaultHugepagesSize: 1G
 pages:
 - count: 2
 node: 0
 size: 1G
```

```
nodeSelector:
 node-role.kubernetes.io/performance: ""
realTimeKernel:
 enabled: true
```

**NOTE**

When you configure reserved and isolated CPUs, the infra containers in pods use the reserved CPUs and the application containers use the isolated CPUs.

## 17.7. MANAGING DEVICE INTERRUPT PROCESSING FOR GUARANTEED POD ISOLATED CPUS

The Node Tuning Operator can manage host CPUs by dividing them into reserved CPUs for cluster and operating system housekeeping duties, including pod infra containers, and isolated CPUs for application containers to run the workloads. This allows you to set CPUs for low latency workloads as isolated.

Device interrupts are load balanced between all isolated and reserved CPUs to avoid CPUs being overloaded, with the exception of CPUs where there is a guaranteed pod running. Guaranteed pod CPUs are prevented from processing device interrupts when the relevant annotations are set for the pod.

In the performance profile, **globallyDisableIrqLoadBalancing** is used to manage whether device interrupts are processed or not. For certain workloads, the reserved CPUs are not always sufficient for dealing with device interrupts, and for this reason, device interrupts are not globally disabled on the isolated CPUs. By default, Node Tuning Operator does not disable device interrupts on isolated CPUs.

### 17.7.1. Finding the effective IRQ affinity setting for a node

Some IRQ controllers lack support for IRQ affinity setting and will always expose all online CPUs as the IRQ mask. These IRQ controllers effectively run on CPU 0.

The following are examples of drivers and hardware that Red Hat are aware lack support for IRQ affinity setting. The list is, by no means, exhaustive:

- Some RAID controller drivers, such as **megaraid\_sas**
- Many non-volatile memory express (NVMe) drivers
- Some LAN on motherboard (LOM) network controllers
- The driver uses **managed\_irqs**

**NOTE**

The reason they do not support IRQ affinity setting might be associated with factors such as the type of processor, the IRQ controller, or the circuitry connections in the motherboard.

If the effective affinity of any IRQ is set to an isolated CPU, it might be a sign of some hardware or driver not supporting IRQ affinity setting. To find the effective affinity, log in to the host and run the following command:

```
$ find /proc/irq -name effective_affinity -printf "%p: " -exec cat {} \;
```

## Example output

```
/proc/irq/0/effective_affinity: 1
/proc/irq/1/effective_affinity: 8
/proc/irq/2/effective_affinity: 0
/proc/irq/3/effective_affinity: 1
/proc/irq/4/effective_affinity: 2
/proc/irq/5/effective_affinity: 1
/proc/irq/6/effective_affinity: 1
/proc/irq/7/effective_affinity: 1
/proc/irq/8/effective_affinity: 1
/proc/irq/9/effective_affinity: 2
/proc/irq/10/effective_affinity: 1
/proc/irq/11/effective_affinity: 1
/proc/irq/12/effective_affinity: 4
/proc/irq/13/effective_affinity: 1
/proc/irq/14/effective_affinity: 1
/proc/irq/15/effective_affinity: 1
/proc/irq/24/effective_affinity: 2
/proc/irq/25/effective_affinity: 4
/proc/irq/26/effective_affinity: 2
/proc/irq/27/effective_affinity: 1
/proc/irq/28/effective_affinity: 8
/proc/irq/29/effective_affinity: 4
/proc/irq/30/effective_affinity: 4
/proc/irq/31/effective_affinity: 8
/proc/irq/32/effective_affinity: 8
/proc/irq/33/effective_affinity: 1
/proc/irq/34/effective_affinity: 2
```

Some drivers use **managed\_irqs**, whose affinity is managed internally by the kernel and userspace cannot change the affinity. In some cases, these IRQs might be assigned to isolated CPUs. For more information about **managed\_irqs**, see [Affinity of managed interrupts cannot be changed even if they target isolated CPU](#).

### 17.7.2. Configuring node interrupt affinity

Configure a cluster node for IRQ dynamic load balancing to control which cores can receive device interrupt requests (IRQ).

#### Prerequisites

- For core isolation, all server hardware components must support IRQ affinity. To check if the hardware components of your server support IRQ affinity, view the server's hardware specifications or contact your hardware provider.

#### Procedure

- Log in to the OpenShift Container Platform cluster as a user with cluster-admin privileges.
- Set the performance profile **apiVersion** to use **performance.openshift.io/v2**.
- Remove the **globallyDisableIrqLoadBalancing** field or set it to **false**.

- Set the appropriate isolated and reserved CPUs. The following snippet illustrates a profile that reserves 2 CPUs. IRQ load-balancing is enabled for pods running on the **isolated** CPU set:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: dynamic-irq-profile
spec:
 cpu:
 isolated: 2-5
 reserved: 0-1
...

```



#### NOTE

When you configure reserved and isolated CPUs, operating system processes, kernel processes, and systemd services run on reserved CPUs. Infrastructure pods run on any CPU except where the low latency workload is running. Low latency workload pods run on exclusive CPUs from the isolated pool. For more information, see "Restricting CPUs for infra and application containers".

## 17.8. CONFIGURING HUGE PAGES

Nodes must pre-allocate huge pages used in an OpenShift Container Platform cluster. Use the Node Tuning Operator to allocate huge pages on a specific node.

OpenShift Container Platform provides a method for creating and allocating huge pages. Node Tuning Operator provides an easier method for doing this using the performance profile.

For example, in the **hugepages pages** section of the performance profile, you can specify multiple blocks of **size**, **count**, and, optionally, **node**:

```
hugepages:
 defaultHugepagesSize: "1G"
 pages:
 - size: "1G"
 count: 4
 node: 0 ①
```

- ① **node** is the NUMA node in which the huge pages are allocated. If you omit **node**, the pages are evenly spread across all NUMA nodes.



#### NOTE

Wait for the relevant machine config pool status that indicates the update is finished.

These are the only configuration steps you need to do to allocate huge pages.

### Verification

- To verify the configuration, see the **/proc/meminfo** file on the node:

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

```
grep -i huge /proc/meminfo
```

### Example output

```
AnonHugePages: ##### ##
ShmemHugePages: 0 kB
HugePages_Total: 2
HugePages_Free: 2
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize: #### ##
Hugetlb: ##### ##
```

- Use **oc describe** to report the new size:

```
$ oc describe node worker-0.ocp4poc.example.com | grep -i huge
```

### Example output

```
hugepages-1g=true
hugepages-##: ##
hugepages-##: ##
```

## 17.8.1. Allocating multiple huge page sizes

You can request huge pages with different sizes under the same container. This allows you to define more complicated pods consisting of containers with different huge page size needs.

For example, you can define sizes **1G** and **2M** and the Node Tuning Operator will configure both sizes on the node, as shown here:

```
spec:
hugepages:
 defaultHugepagesSize: 1G
 pages:
 - count: 1024
 node: 0
 size: 2M
 - count: 4
 node: 1
 size: 1G
```

## 17.9. REDUCING NIC QUEUES USING THE NODE TUNING OPERATOR

The Node Tuning Operator facilitates reducing NIC queues for enhanced performance. Adjustments are made using the performance profile, allowing customization of queues for different network devices.

### 17.9.1. Adjusting the NIC queues with the performance profile

The performance profile lets you adjust the queue count for each network device.

Supported network devices:

- Non-virtual network devices
- Network devices that support multiple queues (channels)

Unsupported network devices:

- Pure software network interfaces
- Block devices
- Intel DPDK virtual functions

## Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- Install the OpenShift CLI (**oc**).

## Procedure

1. Log in to the OpenShift Container Platform cluster running the Node Tuning Operator as a user with **cluster-admin** privileges.
2. Create and apply a performance profile appropriate for your hardware and topology. For guidance on creating a profile, see the "Creating a performance profile" section.
3. Edit this created performance profile:

```
$ oc edit -f <your_profile_name>.yaml
```

4. Populate the **spec** field with the **net** object. The object list can contain two fields:
  - **userLevelNetworking** is a required field specified as a boolean flag. If **userLevelNetworking** is **true**, the queue count is set to the reserved CPU count for all supported devices. The default is **false**.
  - **devices** is an optional field specifying a list of devices that will have the queues set to the reserved CPU count. If the device list is empty, the configuration applies to all network devices. The configuration is as follows:
    - **interfaceName**: This field specifies the interface name, and it supports shell-style wildcards, which can be positive or negative.
      - Example wildcard syntax is as follows: **<string> \***
      - Negative rules are prefixed with an exclamation mark. To apply the net queue changes to all devices other than the excluded list, use **!<device>**, for example, **!eno1**.
    - **vendorID**: The network device vendor ID represented as a 16-bit hexadecimal number with a **0x** prefix.

- **deviceID**: The network device ID (model) represented as a 16-bit hexadecimal number with a **0x** prefix.



### NOTE

When a **deviceID** is specified, the **vendorID** must also be defined. A device that matches all of the device identifiers specified in a device entry **interfaceName**, **vendorID**, or a pair of **vendorID** plus **deviceID** qualifies as a network device. This network device then has its net queues count set to the reserved CPU count.

When two or more devices are specified, the net queues count is set to any net device that matches one of them.

- Set the queue count to the reserved CPU count for all devices by using this example performance profile:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: manual
spec:
 cpu:
 isolated: 3-51,55-103
 reserved: 0-2,52-54
 net:
 userLevelNetworking: true
 nodeSelector:
 node-role.kubernetes.io/worker-cnf: ""
```

- Set the queue count to the reserved CPU count for all devices matching any of the defined device identifiers by using this example performance profile:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: manual
spec:
 cpu:
 isolated: 3-51,55-103
 reserved: 0-2,52-54
 net:
 userLevelNetworking: true
 devices:
 - interfaceName: "eth0"
 - interfaceName: "eth1"
 - vendorID: "0x1af4"
 - deviceID: "0x1000"
 nodeSelector:
 node-role.kubernetes.io/worker-cnf: ""
```

- Set the queue count to the reserved CPU count for all devices starting with the interface name **eth** by using this example performance profile:

```
apiVersion: performance.openshift.io/v2
```

```

kind: PerformanceProfile
metadata:
 name: manual
spec:
 cpu:
 isolated: 3-51,55-103
 reserved: 0-2,52-54
 net:
 userLevelNetworking: true
 devices:
 - interfaceName: "eth*"
 nodeSelector:
 node-role.kubernetes.io/worker-cnf: ""

```

- Set the queue count to the reserved CPU count for all devices with an interface named anything other than **eno1** by using this example performance profile:

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: manual
spec:
 cpu:
 isolated: 3-51,55-103
 reserved: 0-2,52-54
 net:
 userLevelNetworking: true
 devices:
 - interfaceName: "!eno1"
 nodeSelector:
 node-role.kubernetes.io/worker-cnf: ""

```

- Set the queue count to the reserved CPU count for all devices that have an interface name **eth0**, **vendorID** of **0x1af4**, and **deviceId** of **0x1000** by using this example performance profile:

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: manual
spec:
 cpu:
 isolated: 3-51,55-103
 reserved: 0-2,52-54
 net:
 userLevelNetworking: true
 devices:
 - interfaceName: "eth0"
 - vendorID: "0x1af4"
 - deviceId: "0x1000"
 nodeSelector:
 node-role.kubernetes.io/worker-cnf: ""

```

- Apply the updated performance profile:

```
$ oc apply -f <your_profile_name>.yaml
```

## Additional resources

- [Creating a performance profile](#) .

### 17.9.2. Verifying the queue status

In this section, a number of examples illustrate different performance profiles and how to verify the changes are applied.

#### Example 1

In this example, the net queue count is set to the reserved CPU count (2) for *all* supported devices.

The relevant section from the performance profile is:

```
apiVersion: performance.openshift.io/v2
metadata:
 name: performance
spec:
 kind: PerformanceProfile
 spec:
 cpu:
 reserved: 0-1 #total = 2
 isolated: 2-8
 net:
 userLevelNetworking: true
...
```

- Display the status of the queues associated with a device using the following command:



#### NOTE

Run this command on the node where the performance profile was applied.

```
$ ethtool -l <device>
```

- Verify the queue status before the profile is applied:

```
$ ethtool -l ens4
```

#### Example output

```
Channel parameters for ens4:
Pre-set maximums:
RX: 0
TX: 0
Other: 0
Combined: 4
Current hardware settings:
RX: 0
TX: 0
Other: 0
Combined: 4
```

- Verify the queue status after the profile is applied:

```
$ ethtool -l ens4
```

### Example output

```
Channel parameters for ens4:
```

```
Pre-set maximums:
```

```
RX: 0
```

```
TX: 0
```

```
Other: 0
```

```
Combined: 4
```

```
Current hardware settings:
```

```
RX: 0
```

```
TX: 0
```

```
Other: 0
```

```
Combined: 2 1
```

- 1 The combined channel shows that the total count of reserved CPUs for *all* supported devices is 2. This matches what is configured in the performance profile.

### Example 2

In this example, the net queue count is set to the reserved CPU count (2) for *all* supported network devices with a specific **vendorID**.

The relevant section from the performance profile is:

```
apiVersion: performance.openshift.io/v2
metadata:
 name: performance
spec:
 kind: PerformanceProfile
 spec:
 cpu:
 reserved: 0-1 #total = 2
 isolated: 2-8
 net:
 userLevelNetworking: true
 devices:
 - vendorID = 0x1af4
 # ...
```

- Display the status of the queues associated with a device using the following command:



#### NOTE

Run this command on the node where the performance profile was applied.

```
$ ethtool -l <device>
```

- Verify the queue status after the profile is applied:



```
$ ethtool -l ens4
```

### Example output

Channel parameters for ens4:

Pre-set maximums:

RX: 0

TX: 0

Other: 0

Combined: 4

Current hardware settings:

RX: 0

TX: 0

Other: 0

Combined: 2 1

- 1 The total count of reserved CPUs for all supported devices with **vendorID=0x1af4** is 2. For example, if there is another network device **ens2** with **vendorID=0x1af4** it will also have total net queues of 2. This matches what is configured in the performance profile.

### Example 3

In this example, the net queue count is set to the reserved CPU count (2) for *all* supported network devices that match any of the defined device identifiers.

The command **udevadm info** provides a detailed report on a device. In this example the devices are:

```
udevadm info -p /sys/class/net/ens4
...
E: ID_MODEL_ID=0x1000
E: ID_VENDOR_ID=0x1af4
E: INTERFACE=ens4
...
```

```
udevadm info -p /sys/class/net/eth0
...
E: ID_MODEL_ID=0x1002
E: ID_VENDOR_ID=0x1001
E: INTERFACE=eth0
...
```

- Set the net queues to 2 for a device with **interfaceName** equal to **eth0** and any devices that have a **vendorID=0x1af4** with the following performance profile:

```
apiVersion: performance.openshift.io/v2
metadata:
 name: performance
spec:
 kind: PerformanceProfile
 spec:
 cpu:
 reserved: 0-1 #total = 2
 isolated: 2-8
 net:
```

```
userLevelNetworking: true
devices:
- interfaceName = eth0
- vendorID = 0x1af4
...
...
```

- Verify the queue status after the profile is applied:

```
$ ethtool -l ens4
```

### Example output

```
Channel parameters for ens4:
Pre-set maximums:
RX: 0
TX: 0
Other: 0
Combined: 4
Current hardware settings:
RX: 0
TX: 0
Other: 0
Combined: 2 ①
```

- ① The total count of reserved CPUs for all supported devices with **vendorID=0x1af4** is set to 2. For example, if there is another network device **ens2** with **vendorID=0x1af4**, it will also have the total net queues set to 2. Similarly, a device with **interfaceName** equal to **eth0** will have total net queues set to 2.

### 17.9.3. Logging associated with adjusting NIC queues

Log messages detailing the assigned devices are recorded in the respective Tuned daemon logs. The following messages might be recorded to the **/var/log/tuned/tuned.log** file:

- An **INFO** message is recorded detailing the successfully assigned devices:

```
INFO tuned.plugins.base: instance net_test (net): assigning devices ens1, ens2, ens3
```

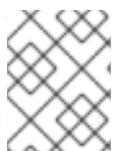
- A **WARNING** message is recorded if none of the devices can be assigned:

```
WARNING tuned.plugins.base: instance net_test: no matching devices available
```

# CHAPTER 18. PROVISIONING REAL-TIME AND LOW LATENCY WORKLOADS

Many organizations need high performance computing and low, predictable latency, especially in the financial and telecommunications industries.

OpenShift Container Platform provides the Node Tuning Operator to implement automatic tuning to achieve low latency performance and consistent response time for OpenShift Container Platform applications. You use the performance profile configuration to make these changes. You can update the kernel to kernel-rt, reserve CPUs for cluster and operating system housekeeping duties, including pod infra containers, isolate CPUs for application containers to run the workloads, and disable unused CPUs to reduce power consumption.



## NOTE

When writing your applications, follow the general recommendations described in [RHEL for Real Time processes and threads](#).

### Additional resources

- [Creating a performance profile](#)

## 18.1. SCHEDULING A LOW LATENCY WORKLOAD ONTO A WORKER WITH REAL-TIME CAPABILITIES

You can schedule low latency workloads onto a worker node where a performance profile that configures real-time capabilities is applied.



## NOTE

To schedule the workload on specific nodes, use label selectors in the **Pod** custom resource (CR). The label selectors must match the nodes that are attached to the machine config pool that was configured for low latency by the Node Tuning Operator.

### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have applied a performance profile in the cluster that tunes worker nodes for low latency workloads.

### Procedure

1. Create a **Pod** CR for the low latency workload and apply it in the cluster, for example:

#### Example Pod spec configured to use real-time processing

```
apiVersion: v1
kind: Pod
metadata:
 name: dynamic-low-latency-pod
```

```

annotations:
 cpu-quota.crio.io: "disable" ①
 cpu-load-balancing.crio.io: "disable" ②
 irq-load-balancing.crio.io: "disable" ③
spec:
 securityContext:
 runAsNonRoot: true
 seccompProfile:
 type: RuntimeDefault
 containers:
 - name: dynamic-low-latency-pod
 image: "registry.redhat.io/openshift4/cnf-tests-rhel8:v4.18"
 command: ["sleep", "10h"]
 resources:
 requests:
 cpu: ②
 memory: "200M"
 limits:
 cpu: ②
 memory: "200M"
 securityContext:
 allowPrivilegeEscalation: false
 capabilities:
 drop: [ALL]
 nodeSelector:
 node-role.kubernetes.io/worker-cnf: "" ④
 runtimeClassName: performance-dynamic-low-latency-profile ⑤
...

```

- ① Disables the CPU completely fair scheduler (CFS) quota at the pod run time.
- ② Disables CPU load balancing.
- ③ Opts the pod out of interrupt handling on the node.
- ④ The **nodeSelector** label must match the label that you specify in the **Node** CR.
- ⑤ **runtimeClassName** must match the name of the performance profile configured in the cluster.

2. Enter the pod **runtimeClassName** in the form `performance-<profile_name>`, where `<profile_name>` is the **name** from the **PerformanceProfile** YAML. In the previous example, the **name** is **performance-dynamic-low-latency-profile**.
3. Ensure the pod is running correctly. Status should be **running**, and the correct cnf-worker node should be set:

```
$ oc get pod -o wide
```

### Expected output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
dynamic-low-latency-pod	1/1	Running	0	5h33m	10.131.0.10	cnf-worker.example.com

- Get the CPUs that the pod configured for IRQ dynamic load balancing runs on:

```
$ oc exec -it dynamic-low-latency-pod -- /bin/bash -c "grep Cpus_allowed_list /proc/self/status | awk '{print $2}'"
```

#### Expected output

```
Cpus_allowed_list: 2-3
```

### Verification

Ensure the node configuration is applied correctly.

- Log in to the node to verify the configuration.

```
$ oc debug node/<node-name>
```

- Verify that you can use the node file system:

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

#### Expected output

```
sh-4.4#
```

- Ensure the default system CPU affinity mask does not include the **dynamic-low-latency-pod** CPUs, for example, CPUs 2 and 3.

```
sh-4.4# cat /proc/irq/default_smp_affinity
```

#### Example output

```
33
```

- Ensure the system IRQs are not configured to run on the **dynamic-low-latency-pod** CPUs:

```
sh-4.4# find /proc/irq/ -name smp_affinity_list -exec sh -c 'i="$1"; mask=$(cat $i); file=$(echo $i); echo $file: $mask' {} \;
```

#### Example output

```
/proc/irq/0/smp_affinity_list: 0-5
/proc/irq/1/smp_affinity_list: 5
/proc/irq/2/smp_affinity_list: 0-5
/proc/irq/3/smp_affinity_list: 0-5
/proc/irq/4/smp_affinity_list: 0
/proc/irq/5/smp_affinity_list: 0-5
/proc/irq/6/smp_affinity_list: 0-5
/proc/irq/7/smp_affinity_list: 0-5
/proc/irq/8/smp_affinity_list: 4
/proc/irq/9/smp_affinity_list: 4
/proc/irq/10/smp_affinity_list: 0-5
```

```
/proc/irq/11/smp_affinity_list: 0
/proc/irq/12/smp_affinity_list: 1
/proc/irq/13/smp_affinity_list: 0-5
/proc/irq/14/smp_affinity_list: 1
/proc/irq/15/smp_affinity_list: 0
/proc/irq/24/smp_affinity_list: 1
/proc/irq/25/smp_affinity_list: 1
/proc/irq/26/smp_affinity_list: 1
/proc/irq/27/smp_affinity_list: 5
/proc/irq/28/smp_affinity_list: 1
/proc/irq/29/smp_affinity_list: 0
/proc/irq/30/smp_affinity_list: 0-5
```



### WARNING

When you tune nodes for low latency, the usage of execution probes in conjunction with applications that require guaranteed CPUs can cause latency spikes. Use other probes, such as a properly configured set of network probes, as an alternative.

## Additional resources

- [Placing pods on specific nodes using node selectors](#)
- [Assigning pods to nodes](#)

## 18.2. CREATING A POD WITH A GUARANTEED QOS CLASS

You can create a pod with a quality of service (QoS) class of **Guaranteed** for high-performance workloads. Configuring a pod with a QoS class of **Guaranteed** ensures that the pod has priority access to the specified CPU and memory resources.

To create a pod with a QoS class of **Guaranteed**, you must apply the following specifications:

- Set identical values for the memory limit and memory request fields for each container in the pod.
- Set identical values for CPU limit and CPU request fields for each container in the pod.

In general, a pod with a QoS class of **Guaranteed** will not be evicted from a node. One exception is during resource contention caused by system daemons exceeding reserved resources. In this scenario, the **kubelet** might evict pods to preserve node stability, starting with the lowest priority pods.

### Prerequisites

- Access to the cluster as a user with the **cluster-admin** role
- The OpenShift CLI (**oc**)

### Procedure

1. Create a namespace for the pod by running the following command:

```
$ oc create namespace qos-example ①
```

- ① This example uses the **qos-example** namespace.

### Example output

```
namespace/qos-example created
```

2. Create the **Pod** resource:

- a. Create a YAML file that defines the **Pod** resource:

#### Example **qos-example.yaml** file

```
apiVersion: v1
kind: Pod
metadata:
 name: qos-demo
 namespace: qos-example
spec:
 securityContext:
 runAsNonRoot: true
 seccompProfile:
 type: RuntimeDefault
 containers:
 - name: qos-demo-ctr
 image: quay.io/openshifttest/hello-openshift:openshift ①
 resources:
 limits:
 memory: "200Mi" ②
 cpu: "1" ③
 requests:
 memory: "200Mi" ④
 cpu: "1" ⑤
 securityContext:
 allowPrivilegeEscalation: false
 capabilities:
 drop: [ALL]
```

- ① This example uses a public **hello-openshift** image.
- ② Sets the memory limit to 200 MB.
- ③ Sets the CPU limit to 1 CPU.
- ④ Sets the memory request to 200 MB.
- ⑤ Sets the CPU request to 1 CPU.

**NOTE**

If you specify a memory limit for a container, but do not specify a memory request, OpenShift Container Platform automatically assigns a memory request that matches the limit. Similarly, if you specify a CPU limit for a container, but do not specify a CPU request, OpenShift Container Platform automatically assigns a CPU request that matches the limit.

- b. Create the **Pod** resource by running the following command:

```
$ oc apply -f qos-example.yaml --namespace=qos-example
```

**Example output**

```
pod/qos-demo created
```

**Verification**

- View the **qosClass** value for the pod by running the following command:

```
$ oc get pod qos-demo --namespace=qos-example --output=yaml | grep qosClass
```

**Example output**

```
qosClass: Guaranteed
```

### 18.3. DISABLING CPU LOAD BALANCING IN A POD

Functionality to disable or enable CPU load balancing is implemented on the CRI-O level. The code under the CRI-O disables or enables CPU load balancing only when the following requirements are met.

- The pod must use the **performance-<profile-name>** runtime class. You can get the proper name by looking at the status of the performance profile, as shown here:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
...
status:
...
runtimeClass: performance-manual
```

The Node Tuning Operator is responsible for the creation of the high-performance runtime handler config snippet under relevant nodes and for creation of the high-performance runtime class under the cluster. It will have the same content as the default runtime handler except that it enables the CPU load balancing configuration functionality.

To disable the CPU load balancing for the pod, the **Pod** specification must include the following fields:

```
apiVersion: v1
kind: Pod
```

```

metadata:
#...
annotations:
#...
cpu-load-balancing.crio.io: "disable"
#...
#...
spec:
#...
runtimeClassName: performance-<profile_name>
#...

```



### NOTE

Only disable CPU load balancing when the CPU manager static policy is enabled and for pods with guaranteed QoS that use whole CPUs. Otherwise, disabling CPU load balancing can affect the performance of other containers in the cluster.

## 18.4. DISABLING POWER SAVING MODE FOR HIGH PRIORITY PODS

You can configure pods to ensure that high priority workloads are unaffected when you configure power saving for the node that the workloads run on.

When you configure a node with a power saving configuration, you must configure high priority workloads with performance configuration at the pod level, which means that the configuration applies to all the cores used by the pod.

By disabling P-states and C-states at the pod level, you can configure high priority workloads for best performance and lowest latency.

**Table 18.1. Configuration for high priority workloads**

Annotation	Possible Values	Description
<b>cpu-c-states.crio.io</b> :	<ul style="list-style-type: none"> <li>• "enable"</li> <li>• "disable"</li> <li>• "max_latency:micro seconds"</li> </ul>	This annotation allows you to enable or disable C-states for each CPU. Alternatively, you can also specify a maximum latency in microseconds for the C-states. For example, enable C-states with a maximum latency of 10 microseconds with the setting <b>cpu-c-states.crio.io: "max_latency:10"</b> . Set the value to " <b>disable</b> " to provide the best performance for a pod.
<b>cpu-freq-governor.cri.o.io:</b>	Any supported <b>cpufreq governor</b> .	Sets the <b>cpufreq</b> governor for each CPU. The " <b>performance</b> " governor is recommended for high priority workloads.

### Prerequisites

- You have configured power saving in the performance profile for the node where the high priority workload pods are scheduled.

### Procedure

1. Add the required annotations to your high priority workload pods. The annotations override the **default** settings.

#### Example high priority workload annotation

```
apiVersion: v1
kind: Pod
metadata:
 #...
 annotations:
 #...
 cpu-c-states.crio.io: "disable"
 cpu-freq-governor.crio.io: "performance"
 #...
 #...
spec:
 #...
 runtimeClassName: performance-<profile_name>
 #...
```

2. Restart the pods to apply the annotation.

#### Additional resources

[Configuring power saving for nodes that run colocated high and low priority workloads](#)

## 18.5. DISABLING CPU CFS QUOTA

To eliminate CPU throttling for pinned pods, create a pod with the **cpu-quota.crio.io: "disable"** annotation. This annotation disables the CPU completely fair scheduler (CFS) quota when the pod runs.

#### Example pod specification with **cpu-quota.crio.io** disabled

```
apiVersion: v1
kind: Pod
metadata:
 annotations:
 cpu-quota.crio.io: "disable"
spec:
 runtimeClassName: performance-<profile_name>
 #...
```



#### NOTE

Only disable CPU CFS quota when the CPU manager static policy is enabled and for pods with guaranteed QoS that use whole CPUs. For example, pods that contain CPU-pinned containers. Otherwise, disabling CPU CFS quota can affect the performance of other containers in the cluster.

#### Additional resources

- [Recommended firmware configuration for vDU cluster hosts](#)

## 18.6. DISABLING INTERRUPT PROCESSING FOR CPUS WHERE PINNED CONTAINERS ARE RUNNING

To achieve low latency for workloads, some containers require that the CPUs they are pinned to do not process device interrupts. A pod annotation, **irq-load-balancing.crio.io**, is used to define whether device interrupts are processed or not on the CPUs where the pinned containers are running. When configured, CRI-O disables device interrupts where the pod containers are running.

To disable interrupt processing for CPUs where containers belonging to individual pods are pinned, ensure that **globallyDisableIrqLoadBalancing** is set to **false** in the performance profile. Then, in the pod specification, set the **irq-load-balancing.crio.io** pod annotation to **disable**.

The following pod specification contains this annotation:

```
apiVersion: performance.openshift.io/v2
kind: Pod
metadata:
 annotations:
 irq-load-balancing.crio.io: "disable"
spec:
 runtimeClassName: performance-<profile_name>
...
...
```

### Additional resources

- [Managing device interrupt processing for guaranteed pod isolated CPUs](#)

# CHAPTER 19. DEBUGGING LOW LATENCY NODE TUNING STATUS

Use the **PerformanceProfile** custom resource (CR) status fields for reporting tuning status and debugging latency issues in the cluster node.

## 19.1. DEBUGGING LOW LATENCY CNF TUNING STATUS

The **PerformanceProfile** custom resource (CR) contains status fields for reporting tuning status and debugging latency degradation issues. These fields report on conditions that describe the state of the operator's reconciliation functionality.

A typical issue can arise when the status of machine config pools that are attached to the performance profile are in a degraded state, causing the **PerformanceProfile** status to degrade. In this case, the machine config pool issues a failure message.

The Node Tuning Operator contains the **performanceProfile.spec.status.Conditions** status field:

```
Status:
Conditions:
 Last Heartbeat Time: 2020-06-02T10:01:24Z
 Last Transition Time: 2020-06-02T10:01:24Z
 Status: True
 Type: Available
 Last Heartbeat Time: 2020-06-02T10:01:24Z
 Last Transition Time: 2020-06-02T10:01:24Z
 Status: True
 Type: Upgradeable
 Last Heartbeat Time: 2020-06-02T10:01:24Z
 Last Transition Time: 2020-06-02T10:01:24Z
 Status: False
 Type: Progressing
 Last Heartbeat Time: 2020-06-02T10:01:24Z
 Last Transition Time: 2020-06-02T10:01:24Z
 Status: False
 Type: Degraded
```

The **Status** field contains **Conditions** that specify **Type** values that indicate the status of the performance profile:

### Available

All machine configs and Tuned profiles have been created successfully and are available for cluster components are responsible to process them (NTO, MCO, Kubelet).

### Upgradeable

Indicates whether the resources maintained by the Operator are in a state that is safe to upgrade.

### Progressing

Indicates that the deployment process from the performance profile has started.

### Degraded

Indicates an error if:

- Validation of the performance profile has failed.

- Creation of all relevant components did not complete successfully.

Each of these types contain the following fields:

### Status

The state for the specific type (**true** or **false**).

### Timestamp

The transaction timestamp.

### Reason string

The machine readable reason.

### Message string

The human readable reason describing the state and error details, if any.

## 19.1.1. Machine config pools

A performance profile and its created products are applied to a node according to an associated machine config pool (MCP). The MCP holds valuable information about the progress of applying the machine configurations created by performance profiles that encompass kernel args, kube config, huge pages allocation, and deployment of rt-kernel. The Performance Profile controller monitors changes in the MCP and updates the performance profile status accordingly.

The only conditions returned by the MCP to the performance profile status is when the MCP is **Degraded**, which leads to **performanceProfile.status.condition.Degraded = true**.

### Example

The following example is for a performance profile with an associated machine config pool (**worker-cnf**) that was created for it:

1. The associated machine config pool is in a degraded state:

```
oc get mcp
```

### Example output

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT		
DEGRADED MACHINECOUNT	AGE			
master	rendered-master-2ee57a93fa6c9181b546ca46e1571d2d	True		False
False	3 3 3 0	2d21h		
worker	rendered-worker-d6b2bdc07d9f5a59a6b68950acf25e5f	True		False
False	2 2 2 0	2d21h		
worker-cnf	rendered-worker-cnf-6c838641b8a08fff08dbd8b02fb63f7c	False		True
True	2 1 1 1	2d20h		

2. The **describe** section of the MCP shows the reason:

```
oc describe mcp worker-cnf
```

### Example output

```

Message: Node node-worker-cnf is reporting: "prepping update:
machineconfig.machineconfiguration.openshift.io \"rendered-worker-cnf-
40b9996919c08e335f3ff230ce1d170\" not
found"
Reason: 1 nodes are reporting degraded status on sync

```

3. The degraded state should also appear under the performance profile **status** field marked as **degraded = true**:

```
oc describe performanceprofiles performance
```

#### Example output

```

Message: Machine config pool worker-cnf Degraded Reason: 1 nodes are reporting
degraded status on sync.
Machine config pool worker-cnf Degraded Message: Node yquinn-q8s5v-w-b-
z5lqn.c.openshift-gce-devel.internal is
reporting: "prepping update: machineconfig.machineconfiguration.openshift.io
\"rendered-worker-cnf-40b9996919c08e335f3ff230ce1d170\" not found". Reason:
MCPDegraded
Status: True
Type: Degraded

```

## 19.2. COLLECTING LOW LATENCY TUNING DEBUGGING DATA FOR RED HAT SUPPORT

When opening a support case, it is helpful to provide debugging information about your cluster to Red Hat Support.

The **must-gather** tool enables you to collect diagnostic information about your OpenShift Container Platform cluster, including node tuning, NUMA topology, and other information needed to debug issues with low latency setup.

For prompt support, supply diagnostic information for both OpenShift Container Platform and low latency tuning.

### 19.2.1. About the must-gather tool

The **oc adm must-gather** CLI command collects the information from your cluster that is most likely needed for debugging issues, such as:

- Resource definitions
- Audit logs
- Service logs

You can specify one or more images when you run the command by including the **--image** argument. When you specify an image, the tool collects data related to that feature or product. When you run **oc adm must-gather**, a new pod is created on the cluster. The data is collected on that pod and saved in a new directory that starts with **must-gather.local**. This directory is created in your current working directory.

## 19.2.2. Gathering low latency tuning data

Use the **oc adm must-gather** CLI command to collect information about your cluster, including features and objects associated with low latency tuning, including:

- The Node Tuning Operator namespaces and child objects.
- **MachineConfigPool** and associated **MachineConfig** objects.
- The Node Tuning Operator and associated Tuned objects.
- Linux kernel command-line options.
- CPU and NUMA topology
- Basic PCI device information and NUMA locality.

### Prerequisites

- Access to the cluster 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.
2. Collect debugging information by running the following command:

```
$ oc adm must-gather
```

### Example output

```
[must-gather] OUT Using must-gather plug-in image: quay.io/openshift-release
When opening a support case, bugzilla, or issue please include the following summary data
along with any other requested information:
ClusterID: 829er0fa-1ad8-4e59-a46e-2644921b7eb6
ClusterVersion: Stable at "<cluster_version>"
ClusterOperators:
All healthy and stable

[must-gather] OUT namespace/openshift-must-gather-8fh4x created
[must-gather] OUT clusterrolebinding.rbac.authorization.k8s.io/must-gather-rhlgc created
[must-gather-5564g] POD 2023-07-17T10:17:37.610340849Z Gathering data for
ns/openshift-cluster-version...
[must-gather-5564g] POD 2023-07-17T10:17:38.786591298Z Gathering data for ns/default...
[must-gather-5564g] POD 2023-07-17T10:17:39.117418660Z Gathering data for
ns/openshift...
[must-gather-5564g] POD 2023-07-17T10:17:39.447592859Z Gathering data for ns/kube-
system...
[must-gather-5564g] POD 2023-07-17T10:17:39.803381143Z Gathering data for
ns/openshift-etcd...
```

```
...
```

**Reprinting Cluster State:**

When opening a support case, bugzilla, or issue please include the following summary data along with any other requested information:

ClusterID: 829er0fa-1ad8-4e59-a46e-2644921b7eb6  
ClusterVersion: Stable at "<cluster\_version>"  
ClusterOperators:  
All healthy and stable

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 directory name created by the **must-gather** tool.

**NOTE**

Create a compressed file to attach the data to a support case or to use with the Performance Profile Creator wrapper script when you create a performance profile.

4. Attach the compressed file to your support case on the [Red Hat Customer Portal](#).

**Additional resources**

- [Gathering data about your cluster with the \*\*must-gather\*\* tool](#)
- [Managing nodes with MachineConfig and KubeletConfig CRs](#)
- [Using the Node Tuning Operator](#)
- [Configuring huge pages at boot time](#)
- [How huge pages are consumed by apps](#)

# CHAPTER 20. PERFORMING LATENCY TESTS FOR PLATFORM VERIFICATION

You can use the Cloud-native Network Functions (CNF) tests image to run latency tests on a CNF-enabled OpenShift Container Platform cluster, where all the components required for running CNF workloads are installed. Run the latency tests to validate node tuning for your workload.

The **cnf-tests** container image is available at [registry.redhat.io/openshift4/cnf-tests-rhel8:v4.18](https://registry.redhat.io/openshift4/cnf-tests-rhel8:v4.18).

## 20.1. PREREQUISITES FOR RUNNING LATENCY TESTS

Your cluster must meet the following requirements before you can run the latency tests:

- You have applied all the required CNF configurations. This includes the **PerformanceProfile** cluster and other configuration according to the reference design specifications (RDS) or your specific requirements.
- You have logged in to **registry.redhat.io** with your Customer Portal credentials by using the **podman login** command.

### Additional resources

- [Scheduling a workload onto a worker with real-time capabilities](#)

## 20.2. MEASURING LATENCY

The **cnf-tests** image uses three tools to measure the latency of the system:

- **hwlatdetect**
- **cyclictest**
- **oslat**

Each tool has a specific use. Use the tools in sequence to achieve reliable test results.

### **hwlatdetect**

Measures the baseline that the bare-metal hardware can achieve. Before proceeding with the next latency test, ensure that the latency reported by **hwlatdetect** meets the required threshold because you cannot fix hardware latency spikes by operating system tuning.

### **cyclictest**

Verifies the real-time kernel scheduler latency after **hwlatdetect** passes validation. The **cyclictest** tool schedules a repeated timer and measures the difference between the desired and the actual trigger times. The difference can uncover basic issues with the tuning caused by interrupts or process priorities. The tool must run on a real-time kernel.

### **oslat**

Behaves similarly to a CPU-intensive DPDK application and measures all the interruptions and disruptions to the busy loop that simulates CPU heavy data processing.

The tests introduce the following environment variables:

**Table 20.1. Latency test environment variables**

Environment variables	Description
<b>LATENCY_TEST_DELAY</b>	Specifies the amount of time in seconds after which the test starts running. You can use the variable to allow the CPU manager reconcile loop to update the default CPU pool. The default value is 0.
<b>LATENCY_TEST_CPUS</b>	Specifies the number of CPUs that the pod running the latency tests uses. If you do not set the variable, the default configuration includes all isolated CPUs.
<b>LATENCY_TEST_RUNTIME</b>	Specifies the amount of time in seconds that the latency test must run. The default value is 300 seconds.
	<p> <b>NOTE</b></p> <p>To prevent the Ginkgo 2.0 test suite from timing out before the latency tests complete, set the <code>-ginkgo.timeout</code> flag to a value greater than <code>LATENCY_TEST_RUNTIME</code> + 2 minutes. If you also set a <code>LATENCY_TEST_DELAY</code> value then you must set <code>-ginkgo.timeout</code> to a value greater than <code>LATENCY_TEST_RUNTIME + LATENCY_TEST_DELAY + 2 minutes</code>. The default timeout value for the Ginkgo 2.0 test suite is 1 hour.</p>
<b>HWLATDETECT_MAXIMUM_LATENCY</b>	Specifies the maximum acceptable hardware latency in microseconds for the workload and operating system. If you do not set the value of <b>HWLATDETECT_MAXIMUM_LATENCY</b> or <b>MAXIMUM_LATENCY</b> , the tool compares the default expected threshold (20µs) and the actual maximum latency in the tool itself. Then, the test fails or succeeds accordingly.
<b>CYCLICTEST_MAXIMUM_LATENCY</b>	Specifies the maximum latency in microseconds that all threads expect before waking up during the <code>cyclictest</code> run. If you do not set the value of <b>CYCLICTEST_MAXIMUM_LATENCY</b> or <b>MAXIMUM_LATENCY</b> , the tool skips the comparison of the expected and the actual maximum latency.
<b>OSLAT_MAXIMUM_LATENCY</b>	Specifies the maximum acceptable latency in microseconds for the <code>oslat</code> test results. If you do not set the value of <b>OSLAT_MAXIMUM_LATENCY</b> or <b>MAXIMUM_LATENCY</b> , the tool skips the comparison of the expected and the actual maximum latency.
<b>MAXIMUM_LATENCY</b>	Unified variable that specifies the maximum acceptable latency in microseconds. Applicable for all available latency tools.



#### NOTE

Variables that are specific to a latency tool take precedence over unified variables. For example, if **OSLAT\_MAXIMUM\_LATENCY** is set to 30 microseconds and **MAXIMUM\_LATENCY** is set to 10 microseconds, the `oslat` test will run with maximum acceptable latency of 30 microseconds.

## 20.3. RUNNING THE LATENCY TESTS

Run the cluster latency tests to validate node tuning for your Cloud-native Network Functions (CNF) workload.



### NOTE

When executing **podman** commands as a non-root or non-privileged user, mounting paths can fail with **permission denied** errors. Depending on your local operating system and SELinux configuration, you might also experience issues running these commands from your home directory. To make the **podman** commands work, run the commands from a folder that is not your home/<username> directory, and append **:Z** to the volumes creation. For example, **-v \$(pwd)/:/kubeconfig:Z**. This allows **podman** to do the proper SELinux relabeling.

This procedure runs the three individual tests **hwlatdetect**, **cyclictest**, and **oslat**. For details on these individual tests, see their individual sections.

### Procedure

1. Open a shell prompt in the directory containing the **kubeconfig** file.

You provide the test image with a **kubeconfig** file in current directory and its related **\$KUBECONFIG** environment variable, mounted through a volume. This allows the running container to use the **kubeconfig** file from inside the container.



### NOTE

In the following command, your local **kubeconfig** is mounted to **kubeconfig/kubeconfig** in the cnf-tests container, which allows access to the cluster.

2. To run the latency tests, run the following command, substituting variable values as appropriate:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUNTIME=600 \
-e MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.18 /usr/bin/test-run.sh \
--ginkgo.v --ginkgo.timeout="24h"
```

The **LATENCY\_TEST\_RUNTIME** is shown in seconds, in this case 600 seconds (10 minutes). The test runs successfully when the maximum observed latency is lower than **MAXIMUM\_LATENCY** (20  $\mu$ s).

If the results exceed the latency threshold, the test fails.

3. Optional: Append **--ginkgo.dry-run** flag to run the latency tests in dry-run mode. This is useful for checking what commands the tests run.
4. Optional: Append **--ginkgo.v** flag to run the tests with increased verbosity.
5. Optional: Append **--ginkgo.timeout="24h"** flag to ensure the Ginkgo 2.0 test suite does not timeout before the latency tests complete.

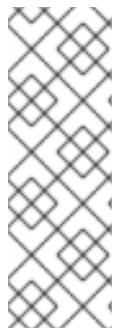


## IMPORTANT

During testing shorter time periods, as shown, can be used to run the tests. However, for final verification and valid results, the test should run for at least 12 hours (43200 seconds).

### 20.3.1. Running **hwlatdetect**

The **hwlatdetect** tool is available in the **rt-kernel** package with a regular subscription of Red Hat Enterprise Linux (RHEL) 9.x.



## NOTE

When executing **podman** commands as a non-root or non-privileged user, mounting paths can fail with **permission denied** errors. Depending on your local operating system and SELinux configuration, you might also experience issues running these commands from your home directory. To make the **podman** commands work, run the commands from a folder that is not your home/<username> directory, and append **:Z** to the volumes creation. For example, **-v \$(pwd)/:/kubeconfig:Z**. This allows **podman** to do the proper SELinux relabeling.

#### Prerequisites

- You have reviewed the prerequisites for running latency tests.

#### Procedure

- To run the **hwlatdetect** tests, run the following command, substituting variable values as appropriate:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUNTIME=600 -e MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.18 \
/usr/bin/test-run.sh --ginkgo.focus="hwlatdetect" --ginkgo.v --ginkgo.timeout="24h"
```

The **hwlatdetect** test runs for 10 minutes (600 seconds). The test runs successfully when the maximum observed latency is lower than **MAXIMUM\_LATENCY** (20  $\mu$ s).

If the results exceed the latency threshold, the test fails.



## IMPORTANT

During testing shorter time periods, as shown, can be used to run the tests. However, for final verification and valid results, the test should run for at least 12 hours (43200 seconds).

#### Example failure output

```
running /usr/bin/cnftests -ginkgo.v -ginkgo.focus=hwlatdetect
I0908 15:25:20.023712 27 request.go:601] Waited for 1.046586367s due to client-side
throttling, not priority and fairness, request:
GET:https://api.hxcl6.lab.eng.tlv2.redhat.com:6443/apis/imageregistry.operator.openshift.io/v1?
timeout=32s
Running Suite: CNF Features e2e integration tests
```

```
=====
Random Seed: 1662650718
Will run 1 of 3 specs

[...]

• Failure [283.574 seconds]
[performance] Latency Test
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:62
with the hwlatdetect image
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:228
should succeed [It]
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:236

Log file created at: 2022/09/08 15:25:27
Running on machine: hwlatdetect-b6n4n
Binary: Built with gc go1.17.12 for linux/amd64
Log line format: [IWEF]mmdd hh:mm:ss.uuuuuu threadid file:line] msg
I0908 15:25:27.160620 1 node.go:39] Environment information: /proc/cmdline:
BOOT_IMAGE=(hd1,gpt3)/ostree/rhcos-
c6491e1eedf6c1f12ef7b95e14ee720bf48359750ac900b7863c625769ef5fb9/vmlinuz-4.18.0-
372.19.1.el8_6.x86_64 random.trust_cpu=on console=tty0 console=ttyS0,115200n8
ignition.platform.id=metal
ostree=/ostree/boot.1/rhcos/c6491e1eedf6c1f12ef7b95e14ee720bf48359750ac900b7863c625
769ef5fb9/0 ip=dhcp root=UUID=5f80c283-f6e6-4a27-9b47-a287157483b2 rw
rootflags=prjquota boot=UUID=773bf59a-baf4-48fc-9a87-f62252d739d3 skew_tick=1
nohz=on rcu_nocbs=0-3 tuned.non_isolcpus=0000ffff,ffffffff,fffffff0
systemd.cpu_affinity=4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29
,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,
60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79 intel_iommu=on iommu=pt
isolcpus=managed_irq,0-3 nohz_full=0-3 tsc=nosoftlockup nmi_watchdog=0
mce=off skew_tick=1 rcutree.kthread_prio=11 ++
I0908 15:25:27.160830 1 node.go:46] Environment information: kernel version 4.18.0-
372.19.1.el8_6.x86_64
I0908 15:25:27.160857 1 main.go:50] running the hwlatdetect command with
arguments [/usr/bin/hwlatdetect --threshold 1 --hardlimit 1 --duration 100 --window
10000000us --width 950000us]
F0908 15:27:10.603523 1 main.go:53] failed to run hwlatdetect command; out:
hwlatdetect: test duration 100 seconds
detector: tracer
parameters:
 Latency threshold: 1us 1
 Sample window: 10000000us
 Sample width: 950000us
 Non-sampling period: 9050000us
 Output File: None

Starting test
test finished
Max Latency: 326us 2
Samples recorded: 5
Samples exceeding threshold: 5
ts: 1662650739.017274507, inner:6, outer:6
```

```
ts: 1662650749.257272414, inner:14, outer:326
ts: 1662650779.977272835, inner:314, outer:12
ts: 1662650800.457272384, inner:3, outer:9
ts: 1662650810.697273520, inner:3, outer:2
```

[...]

JUnit report was created: /junit.xml/cnftests-junit.xml

Summarizing 1 Failure:

```
[Fail] [performance] Latency Test with the hwlatdetect image [It] should succeed
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:476
```

```
Ran 1 of 194 Specs in 365.797 seconds
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 2 Skipped
--- FAIL: TestTest (366.08s)
FAIL
```

- 1 You can configure the latency threshold by using the **MAXIMUM\_LATENCY** or the **HWLATDETECT\_MAXIMUM\_LATENCY** environment variables.
- 2 The maximum latency value measured during the test.

### Example hwlatdetect test results

You can capture the following types of results:

- Rough results that are gathered after each run to create a history of impact on any changes made throughout the test.
- The combined set of the rough tests with the best results and configuration settings.

### Example of good results

```
hwlatdetect: test duration 3600 seconds
detector: tracer
parameters:
Latency threshold: 10us
Sample window: 1000000us
Sample width: 950000us
Non-sampling period: 50000us
Output File: None
```

```
Starting test
test finished
Max Latency: Below threshold
Samples recorded: 0
```

The **hwlatdetect** tool only provides output if the sample exceeds the specified threshold.

### Example of bad results

```

hwlatdetect: test duration 3600 seconds
detector: tracer
parameters:Latency threshold: 10usSample window: 1000000us
Sample width: 950000usNon-sampling period: 50000usOutput File: None

Starting tests:1610542421.275784439, inner:78, outer:81
ts: 1610542444.330561619, inner:27, outer:28
ts: 1610542445.332549975, inner:39, outer:38
ts: 1610542541.568546097, inner:47, outer:32
ts: 1610542590.681548531, inner:13, outer:17
ts: 1610543033.818801482, inner:29, outer:30
ts: 1610543080.938801990, inner:90, outer:76
ts: 1610543129.065549639, inner:28, outer:39
ts: 1610543474.859552115, inner:28, outer:35
ts: 1610543523.973856571, inner:52, outer:49
ts: 1610543572.089799738, inner:27, outer:30
ts: 1610543573.091550771, inner:34, outer:28
ts: 1610543574.093555202, inner:116, outer:63

```

The output of **hwlatdetect** shows that multiple samples exceed the threshold. However, the same output can indicate different results based on the following factors:

- The duration of the test
- The number of CPU cores
- The host firmware settings



#### WARNING

Before proceeding with the next latency test, ensure that the latency reported by **hwlatdetect** meets the required threshold. Fixing latencies introduced by hardware might require you to contact the system vendor support.

Not all latency spikes are hardware related. Ensure that you tune the host firmware to meet your workload requirements. For more information, see [Setting firmware parameters for system tuning](#).

#### 20.3.2. Running `cyclictest`

The **cyclictest** tool measures the real-time kernel scheduler latency on the specified CPUs.



## NOTE

When executing **podman** commands as a non-root or non-privileged user, mounting paths can fail with **permission denied** errors. Depending on your local operating system and SELinux configuration, you might also experience issues running these commands from your home directory. To make the **podman** commands work, run the commands from a folder that is not your home/<username> directory, and append **:Z** to the volumes creation. For example, **-v \$(pwd)/:/kubeconfig:Z**. This allows **podman** to do the proper SELinux relabeling.

## Prerequisites

- You have reviewed the prerequisites for running latency tests.

## Procedure

- To perform the **cyclitest**, run the following command, substituting variable values as appropriate:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_CPUS=10 -e LATENCY_TEST_RUNTIME=600 -e
MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.18 \
/usr/bin/test-run.sh --ginkgo.focus="cyclitest" --ginkgo.v --ginkgo.timeout="24h"
```

The command runs the **cyclitest** tool for 10 minutes (600 seconds). The test runs successfully when the maximum observed latency is lower than **MAXIMUM\_LATENCY** (in this example, 20 µs). Latency spikes of 20 µs and above are generally not acceptable for telco RAN workloads.

If the results exceed the latency threshold, the test fails.



## IMPORTANT

During testing shorter time periods, as shown, can be used to run the tests. However, for final verification and valid results, the test should run for at least 12 hours (43200 seconds).

## Example failure output

```
running /usr/bin/cnftests -ginkgo.v -ginkgo.focus=cyclitest
I0908 13:01:59.193776 27 request.go:601] Waited for 1.046228824s due to client-side
throttling, not priority and fairness, request: GET:https://api.compute-
1.example.com:6443/apis/packages.coreos.com/v1?timeout=32s
Running Suite: CNF Features e2e integration tests
=====
Random Seed: 1662642118
Will run 1 of 3 specs

[...]

Summarizing 1 Failure:

[Fail] [performance] Latency Test with the cyclitest image [It] should succeed
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
```

```
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:220
```

```
Ran 1 of 194 Specs in 161.151 seconds
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 2 Skipped
--- FAIL: TestTest (161.48s)
FAIL
```

### Example cyclictest results

The same output can indicate different results for different workloads. For example, spikes up to 18μs are acceptable for 4G DU workloads, but not for 5G DU workloads.

### Example of good results

```
running cmd: cyclictest -q -D 10m -p 1 -t 16 -a 2,4,6,8,10,12,14,16,54,56,58,60,62,64,66,68 -h 30 -i
1000 -m
Histogram
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 000000 000000
000001 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 000000 000000
000002 579506 535967 418614 573648 532870 529897 489306 558076 582350 585188
583793 223781 532480 569130 472250 576043
More histogram entries ...
Total: 000600000 000600000 000600000 000599999 000599999 000599999 000599998
000599998 000599998 000599997 000599997 000599996 000599996 000599995 000599995
000599995
Min Latencies: 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002
00002 00002 00002 00002
Avg Latencies: 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002
00002 00002 00002 00002
Max Latencies: 00005 00005 00004 00005 00004 00004 00005 00005 00006 00005 00004 00005
00004 00004 00005 00004
Histogram Overflows: 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000
00000 00000 00000 00000
Histogram Overflow at cycle number:
Thread 0:
Thread 1:
Thread 2:
Thread 3:
Thread 4:
Thread 5:
Thread 6:
Thread 7:
Thread 8:
Thread 9:
Thread 10:
Thread 11:
Thread 12:
Thread 13:
Thread 14:
Thread 15:
```

### Example of bad results

```
running cmd: cyclictest -q -D 10m -p 1 -t 16 -a 2,4,6,8,10,12,14,16,54,56,58,60,62,64,66,68 -h 30 -i
```

```

1000 -m
Histogram
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
000001 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
000002 564632 579686 354911 563036 492543 521983 515884 378266 592621 463547
482764 591976 590409 588145 589556 353518
More histogram entries ...
Total: 000599999 000599999 000599999 000599997 000599997 000599998 000599998
000599997 000599997 000599996 000599995 000599996 000599995 000599995 000599995
000599993
Min Latencies: 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002
00002 00002 00002 00002
Avg Latencies: 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002 00002
00002 00002 00002 00002
Max Latencies: 00493 00387 00271 00619 00541 00513 00009 00389 00252 00215 00539 00498
00363 00204 00068 00520
Histogram Overflows: 00001 00001 00001 00002 00002 00001 00000 00001 00001 00001 00002
00001 00001 00001 00001 00002
Histogram Overflow at cycle number:
Thread 0: 155922
Thread 1: 110064
Thread 2: 110064
Thread 3: 110063 155921
Thread 4: 110063 155921
Thread 5: 155920
Thread 6:
Thread 7: 110062
Thread 8: 110062
Thread 9: 155919
Thread 10: 110061 155919
Thread 11: 155918
Thread 12: 155918
Thread 13: 110060
Thread 14: 110060
Thread 15: 110059 155917

```

### 20.3.3. Running oslat

The **oslat** test simulates a CPU-intensive DPDK application and measures all the interruptions and disruptions to test how the cluster handles CPU heavy data processing.



#### NOTE

When executing **podman** commands as a non-root or non-privileged user, mounting paths can fail with **permission denied** errors. Depending on your local operating system and SELinux configuration, you might also experience issues running these commands from your home directory. To make the **podman** commands work, run the commands from a folder that is not your home/<username> directory, and append **:Z** to the volumes creation. For example, **-v \$(pwd)/:/kubeconfig:Z**. This allows **podman** to do the proper SELinux relabeling.

#### Prerequisites

- You have reviewed the prerequisites for running latency tests.

## Procedure

- To perform the **oslat** test, run the following command, substituting variable values as appropriate:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_CPUS=10 -e LATENCY_TEST_RUNTIME=600 -e
MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.18 \
/usr/bin/test-run.sh --ginkgo.focus="oslat" --ginkgo.v --ginkgo.timeout="24h"
```

**LATENCY\_TEST\_CPUS** specifies the number of CPUs to test with the **oslat** command.

The command runs the **oslat** tool for 10 minutes (600 seconds). The test runs successfully when the maximum observed latency is lower than **MAXIMUM\_LATENCY** (20 µs).

If the results exceed the latency threshold, the test fails.



### IMPORTANT

During testing shorter time periods, as shown, can be used to run the tests. However, for final verification and valid results, the test should run for at least 12 hours (43200 seconds).

## Example failure output

```
running /usr/bin/cnftests -ginkgo.v -ginkgo.focus=oslat
I0908 12:51:55.999393 27 request.go:601] Waited for 1.044848101s due to client-side
throttling, not priority and fairness, request: GET:https://compute-
1.example.com:6443/apis/machineconfiguration.openshift.io/v1?timeout=32s
Running Suite: CNF Features e2e integration tests
=====
Random Seed: 1662641514
Will run 1 of 3 specs

[...]
• Failure [77.833 seconds]
[performance] Latency Test
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:62
with the oslat image
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:128
should succeed [It]
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:153
```

The current latency 304 is bigger than the expected one 1 : **1**

[...]

Summarizing 1 Failure:

```
[Fail] [performance] Latency Test with the oslat image [It] should succeed
/remote-source/app/vendor/github.com/openshift/cluster-node-tuning-
operator/test/e2e/performanceprofile/functests/4_latency/latency.go:177
```

```
Ran 1 of 194 Specs in 161.091 seconds
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 2 Skipped
--- FAIL: TestTest (161.42s)
FAIL
```

- 1 In this example, the measured latency is outside the maximum allowed value.

## 20.4. GENERATING A LATENCY TEST FAILURE REPORT

Use the following procedures to generate a JUnit latency test output and test failure report.

### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.

### Procedure

- Create a test failure report with information about the cluster state and resources for troubleshooting by passing the **--report** parameter with the path to where the report is dumped:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -v $(pwd)/reportdest:<report_folder_path> \
-e KUBECONFIG=/kubeconfig/kubeconfig registry.redhat.io/openshift4/cnf-tests-rhel8:v4.18 \
/usr/bin/test-run.sh --report <report_folder_path> --ginkgo.v
```

where:

**<report\_folder\_path>**

Is the path to the folder where the report is generated.

## 20.5. GENERATING A JUNIT LATENCY TEST REPORT

Use the following procedures to generate a JUnit latency test output and test failure report.

### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.

### Procedure

- Create a JUnit-compliant XML report by passing the **--junit** parameter together with the path to where the report is dumped:

**NOTE**

You must create the **junit** folder before running this command.

```
$ podman run -v $(pwd)/:/kubeconfig:Z -v $(pwd)/junit:/junit \
-e KUBECONFIG=/kubeconfig/kubeconfig registry.redhat.io/openshift4/cnf-tests-rhel8:v4.18 \
/usr/bin/test-run.sh --ginkgo.junit-report junit/<file-name>.xml --ginkgo.v
```

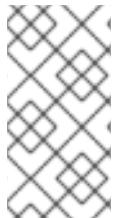
where:

**junit**

Is the folder where the junit report is stored.

## 20.6. RUNNING LATENCY TESTS ON A SINGLE-NODE OPENSHIFT CLUSTER

You can run latency tests on single-node OpenShift clusters.

**NOTE**

When executing **podman** commands as a non-root or non-privileged user, mounting paths can fail with **permission denied** errors. To make the **podman** command work, append **:Z** to the volumes creation; for example, **-v \$(pwd)/:/kubeconfig:Z**. This allows **podman** to do the proper SELinux relabeling.

### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have applied a cluster performance profile by using the Node Tuning Operator.

### Procedure

- To run the latency tests on a single-node OpenShift cluster, run the following command:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUNTIME=<time_in_seconds> registry.redhat.io/openshift4/cnf-tests-rhel8:v4.18 \
/usr/bin/test-run.sh --ginkgo.v --ginkgo.timeout="24h"
```

**NOTE**

The default runtime for each test is 300 seconds. For valid latency test results, run the tests for at least 12 hours by updating the **LATENCY\_TEST\_RUNTIME** variable. To run the buckets latency validation step, you must specify a maximum latency. For details on maximum latency variables, see the table in the "Measuring latency" section.

After running the test suite, all the dangling resources are cleaned up.

## 20.7. RUNNING LATENCY TESTS IN A DISCONNECTED CLUSTER

The CNF tests image can run tests in a disconnected cluster that is not able to reach external registries. This requires two steps:

1. Mirroring the **cnf-tests** image to the custom disconnected registry.
2. Instructing the tests to consume the images from the custom disconnected registry.

### Mirroring the images to a custom registry accessible from the cluster

A **mirror** executable is shipped in the image to provide the input required by **oc** to mirror the test image to a local registry.

1. Run this command from an intermediate machine that has access to the cluster and [registry.redhat.io](#):

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.18 \
/usr/bin/mirror -registry <disconnected_registry> | oc image mirror -f -
```

where:

**<disconnected\_registry>**

Is the disconnected mirror registry you have configured, for example, **my.local.registry:5000/**.

2. When you have mirrored the **cnf-tests** image into the disconnected registry, you must override the original registry used to fetch the images when running the tests, for example:

```
podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e IMAGE_REGISTRY=<disconnected_registry> \
-e CNF_TESTS_IMAGE="cnf-tests-rhel8:v4.18" \
-e LATENCY_TEST_RUNTIME=<time_in_seconds> \
<disconnected_registry>/cnf-tests-rhel8:v4.18 /usr/bin/test-run.sh --ginkgo.v --
ginkgo.timeout="24h"
```

### Configuring the tests to consume images from a custom registry

You can run the latency tests using a custom test image and image registry using **CNF\_TESTS\_IMAGE** and **IMAGE\_REGISTRY** variables.

- To configure the latency tests to use a custom test image and image registry, run the following command:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e IMAGE_REGISTRY=<custom_image_registry> \
-e CNF_TESTS_IMAGE=<custom_cnf-tests_image> \
-e LATENCY_TEST_RUNTIME=<time_in_seconds> \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.18 /usr/bin/test-run.sh --ginkgo.v --
ginkgo.timeout="24h"
```

where:

**<custom\_image\_registry>**

is the custom image registry, for example, **custom.registry:5000/**.

## &lt;custom\_cnf-tests\_image&gt;

is the custom cnf-tests image, for example, **custom-cnf-tests-image:latest**.

**Mirroring images to the cluster OpenShift image registry**

OpenShift Container Platform provides a built-in container image registry, which runs as a standard workload on the cluster.

**Procedure**

1. Gain external access to the registry by exposing it with a route:

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster --patch '{"spec": {"defaultRoute":true}}' --type=merge
```

2. Fetch the registry endpoint by running the following command:

```
$ REGISTRY=$(oc get route default-route -n openshift-image-registry --template='{{ .spec.host }}')
```

3. Create a namespace for exposing the images:

```
$ oc create ns cnftests
```

4. Make the image stream available to all the namespaces used for tests. This is required to allow the tests namespaces to fetch the images from the **cnf-tests** image stream. Run the following commands:

```
$ oc policy add-role-to-user system:image-puller system:serviceaccount:cnf-features-testing:default --namespace=cnftests
```

```
$ oc policy add-role-to-user system:image-puller system:serviceaccount:performance-addon-operators-testing:default --namespace=cnftests
```

5. Retrieve the docker secret name and auth token by running the following commands:

```
$ SECRET=$(oc -n cnftests get secret | grep builder-docker | awk {'print $1'})
```

```
$ TOKEN=$(oc -n cnftests get secret $SECRET -o jsonpath=".data[\".dockercfg\"]" | base64 --decode | jq '.["image-registry.openshift-image-registry.svc:5000"].auth')
```

6. Create a **dockerauth.json** file, for example:

```
$ echo "{\"auths\": { \"$REGISTRY\": { \"auth\": \"$TOKEN\" } }}" > dockerauth.json
```

7. Do the image mirroring:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
registry.redhat.io/openshift4/cnf-tests-rhel8:4.18 \
/usr/bin/mirror -registry $REGISTRY/cnftests | oc image mirror --insecure=true \
-a=$(pwd)/dockerauth.json -f -
```

8. Run the tests:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
-e LATENCY_TEST_RUNTIME=<time_in_seconds> \
-e IMAGE_REGISTRY=image-registry.openshift-image-registry.svc:5000/cnftests cnf-tests-local:latest /usr/bin/test-run.sh --ginkgo.v --ginkgo.timeout="24h"
```

## Mirroring a different set of test images

You can optionally change the default upstream images that are mirrored for the latency tests.

### Procedure

1. The **mirror** command tries to mirror the upstream images by default. This can be overridden by passing a file with the following format to the image:

```
[
 {
 "registry": "public.registry.io:5000",
 "image": "imageforcnftests:4.18"
 }
]
```

2. Pass the file to the **mirror** command, for example saving it locally as **images.json**. With the following command, the local path is mounted in **/kubeconfig** inside the container and that can be passed to the mirror command.

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.18 /usr/bin/mirror \
--registry "my.local.registry:5000/" --images "/kubeconfig/images.json" \
| oc image mirror -f -
```

## 20.8. TROUBLESHOOTING ERRORS WITH THE CNF-TESTS CONTAINER

To run latency tests, the cluster must be accessible from within the **cnf-tests** container.

### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.

### Procedure

- Verify that the cluster is accessible from inside the **cnf-tests** container by running the following command:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.18 \
oc get nodes
```

If this command does not work, an error related to spanning across DNS, MTU size, or firewall access might be occurring.

# CHAPTER 21. IMPROVING CLUSTER STABILITY IN HIGH LATENCY ENVIRONMENTS USING WORKER LATENCY PROFILES

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

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

1. The node controller on the control plane updates the node health to **Unhealthy** and marks the node **Ready** condition `Unknown`.
2. In response, the scheduler stops scheduling pods to that node.
3. The Node Lifecycle Controller adds a **node.kubernetes.io/unreachable** taint with a **NoExecute** effect to the node and schedules any pods on the node for eviction after five minutes, by default.

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

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

These worker latency profiles contain three sets of parameters that are predefined with carefully tuned values to control the reaction of the cluster to increased latency. There is no need to experimentally find the best values manually.

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

## 21.1. UNDERSTANDING WORKER LATENCY PROFILES

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



### IMPORTANT

Setting these parameters manually is not supported. Incorrect parameter settings adversely affect cluster stability.

All worker latency profiles configure the following parameters:

#### **node-status-update-frequency**

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

#### **node-monitor-grace-period**

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

#### **default-not-ready-toleration-seconds**

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

#### **default-unreachable-toleration-seconds**

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

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

- The Machine Config Operator (MCO) updates the **node-status-update-frequency** parameter on the worker nodes.
- The Kubernetes Controller Manager updates the **node-monitor-grace-period** parameter on the control plane nodes.
- The Kubernetes API Server Operator updates the **default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds** parameters on the control plane nodes.

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	<b>node-status-update-frequency</b>	10s
	Kubelet Controller Manager	<b>node-monitor-grace-period</b>	40s

Profile	Component	Parameter	Value
	Kubernetes API Server Operator	<b>default-not-ready-toleration-seconds</b>	300s
	Kubernetes API Server Operator	<b>default-unreachable-toleration-seconds</b>	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	<b>node-status-update-frequency</b>	20s
	Kubelet Controller Manager	<b>node-monitor-grace-period</b>	2m
	Kubernetes API Server Operator	<b>default-not-ready-toleration-seconds</b>	60s
	Kubernetes API Server Operator	<b>default-unreachable-toleration-seconds</b>	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	<b>node-status-update-frequency</b>	1m
	Kubelet Controller Manager	<b>node-monitor-grace-period</b>	5m
	Kubernetes API Server Operator	<b>default-not-ready-toleration-seconds</b>	60s
	Kubernetes API Server Operator	<b>default-unreachable-toleration-seconds</b>	60s



#### NOTE

The latency profiles do not support custom machine config pools, only the default worker machine config pools.

## 21.2. IMPLEMENTING WORKER LATENCY PROFILES AT CLUSTER CREATION



#### IMPORTANT

To edit the configuration of the installation program, first use the command **openshift-install create manifests** to create the default node manifest and other manifest YAML files. This file structure must exist before you can add **workerLatencyProfile**. The platform on which you are installing might have varying requirements. Refer to the Installing section of the documentation for your specific platform.

The **workerLatencyProfile** must be added to the manifest in the following sequence:

1. Create the manifest needed to build the cluster, using a folder name appropriate for your installation.
2. Create a YAML file to define **config.node**. The file must be in the **manifests** directory.
3. When defining **workerLatencyProfile** in the manifest for the first time, specify any of the profiles at cluster creation time: **Default**, **MediumUpdateAverageReaction** or **LowUpdateSlowReaction**.

#### Verification

- Here is an example manifest creation showing the **spec.workerLatencyProfile Default** value in the manifest file:

```
$ openshift-install create manifests --dir=<cluster-install-dir>
```

- Edit the manifest and add the value. In this example we use **vi** to show an example manifest file with the "Default" **workerLatencyProfile** value added:

```
$ vi <cluster-install-dir>/manifests/config-node-default-profile.yaml
```

#### Example output

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
 name: cluster
spec:
 workerLatencyProfile: "Default"
```

### 21.3. USING AND CHANGING WORKER LATENCY PROFILES

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

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



#### NOTE

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

#### Procedure

To move from the default worker latency profile:

1. Move to the medium worker latency profile:

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

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

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

#### Example node.config object

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
 annotations:
 include.release.openshift.io/ibm-cloud-managed: "true"
 include.release.openshift.io/self-managed-high-availability: "true"
 include.release.openshift.io/single-node-developer: "true"
 release.openshift.io/create-only: "true"
 creationTimestamp: "2022-07-08T16:02:51Z"
```

```

generation: 1
name: cluster
ownerReferences:
- apiVersion: config.openshift.io/v1
 kind: ClusterVersion
 name: version
 uid: 36282574-bf9f-409e-a6cd-3032939293eb
resourceVersion: "1865"
uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
 workerLatencyProfile: MediumUpdateAverageReaction ①

...

```

- ① Specifies the medium worker latency policy.

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

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

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

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

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

#### Example **node.config** object

```

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

...

```

- ① Specifies use of the low worker latency policy.

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

### Verification

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

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

### Example output

```
...
- lastTransitionTime: "2022-07-11T19:47:10Z"
 reason: ProfileUpdated
 status: "False"
 type: WorkerLatencyProfileProgressing
- lastTransitionTime: "2022-07-11T19:47:10Z" 1
 message: all static pod revision(s) have updated latency profile
 reason: ProfileUpdated
 status: "True"
 type: WorkerLatencyProfileComplete
- lastTransitionTime: "2022-07-11T19:20:11Z"
 reason: AsExpected
 status: "False"
 type: WorkerLatencyProfileDegraded
- lastTransitionTime: "2022-07-11T19:20:36Z"
 status: "False"
...
```

- 1** Specifies that the profile is applied and active.

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

## 21.4. EXAMPLE STEPS FOR DISPLAYING RESULTING VALUES OF WORKERLATENCYPROFILE

You can display the values in the **workerLatencyProfile** with the following commands.

### Verification

- Check the **default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds** fields output by the Kube API Server:

```
$ oc get KubeAPIServer -o yaml | grep -A 1 default-
```

### Example output

```
default-not-ready-toleration-seconds:
- "300"
default-unreachable-toleration-seconds:
- "300"
```

2. Check the values of the **node-monitor-grace-period** field from the Kube Controller Manager:

```
$ oc get KubeControllerManager -o yaml | grep -A 1 node-monitor
```

#### Example output

```
node-monitor-grace-period:
- 40s
```

3. Check the **nodeStatusUpdateFrequency** value from the Kubelet. Set the directory **/host** as the root directory within the debug shell. By changing the root directory to **/host**, you can run binaries contained in the host's executable paths:

```
$ oc debug node/<worker-node-name>
$ chroot /host
cat /etc/kubernetes/kubelet.conf|grep nodeStatusUpdateFrequency
```

#### Example output

```
"nodeStatusUpdateFrequency": "10s"
```

These outputs validate the set of timing variables for the Worker Latency Profile.

# CHAPTER 22. WORKLOAD PARTITIONING

Workload partitioning separates compute node CPU resources into distinct CPU sets. The primary objective is to keep platform pods on the specified cores to avoid interrupting the CPUs the customer workloads are running on.

Workload partitioning isolates OpenShift Container Platform services, cluster management workloads, and infrastructure pods to run on a reserved set of CPUs. This ensures that the remaining CPUs in the cluster deployment are untouched and available exclusively for non-platform workloads. The minimum number of reserved CPUs required for the cluster management is four CPU Hyper-Threads (HTs).

In the context of enabling workload partitioning and managing CPU resources effectively, nodes that are not configured correctly will not be permitted to join the cluster through a node admission webhook. When the workload partitioning feature is enabled, the machine config pools for control plane and worker will be supplied with configurations for nodes to use. Adding new nodes to these pools will make sure they are correctly configured before joining the cluster.

Currently, nodes must have uniform configurations per machine config pool to ensure that correct CPU affinity is set across all nodes within that pool. After admission, nodes within the cluster identify themselves as supporting a new resource type called **management.workload.openshift.io/cores** and accurately report their CPU capacity. Workload partitioning can be enabled during cluster installation only by adding the additional field **cpuPartitioningMode** to the **install-config.yaml** file.

When workload partitioning is enabled, the **management.workload.openshift.io/cores** resource allows the scheduler to correctly assign pods based on the **cpushares** capacity of the host, not just the default **cpuset**. This ensures more precise allocation of resources for workload partitioning scenarios.

Workload partitioning ensures that CPU requests and limits specified in the pod's configuration are respected. In OpenShift Container Platform 4.16 or later, accurate CPU usage limits are set for platform pods through CPU partitioning. As workload partitioning uses the custom resource type of **management.workload.openshift.io/cores**, the values for requests and limits are the same due to a requirement by Kubernetes for extended resources. However, the annotations modified by workload partitioning correctly reflect the desired limits.



## NOTE

Extended resources cannot be overcommitted, so request and limit must be equal if both are present in a container spec.

## 22.1. ENABLING WORKLOAD PARTITIONING

With workload partitioning, cluster management pods are annotated to correctly partition them into a specified CPU affinity. These pods operate normally within the minimum size CPU configuration specified by the reserved value in the Performance Profile. Additional Day 2 Operators that make use of workload partitioning should be taken into account when calculating how many reserved CPU cores should be set aside for the platform.

Workload partitioning isolates user workloads from platform workloads using standard Kubernetes scheduling capabilities.



## NOTE

You can enable workload partitioning during cluster installation only. You cannot disable workload partitioning postinstallation. However, you can change the CPU configuration for **reserved** and **isolated** CPUs postinstallation.

Use this procedure to enable workload partitioning cluster wide:

### Procedure

- In the **install-config.yaml** file, add the additional field **cpuPartitioningMode** and set it to **AllNodes**.

```
apiVersion: v1
baseDomain: devcluster.openshift.com
cpuPartitioningMode: AllNodes 1
compute:
- architecture: amd64
 hyperthreading: Enabled
 name: worker
 platform: {}
 replicas: 3
controlPlane:
 architecture: amd64
 hyperthreading: Enabled
 name: master
 platform: {}
 replicas: 3
```

- 1** Sets up a cluster for CPU partitioning at install time. The default value is **None**.

## 22.2. PERFORMANCE PROFILES AND WORKLOAD PARTITIONING

Applying a performance profile allows you to make use of the workload partitioning feature. An appropriately configured performance profile specifies the **isolated** and **reserved** CPUs. The recommended way to create a performance profile is to use the Performance Profile Creator (PPC) tool to create the performance profile.

### Additional resources

- [About the Performance Profile Creator](#)

## 22.3. SAMPLE PERFORMANCE PROFILE CONFIGURATION

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 # if you change this name make sure the 'include' line in TunedPerformancePatch.yaml
 # matches this name: include=openshift-node-performance-${PerformanceProfile.metadata.name}
 # Also in file 'validatorCRs/informDuValidator.yaml':
 # name: 50-performance-${PerformanceProfile.metadata.name}
 name: openshift-node-performance-profile
 annotations:
 ran.openshift.io/reference-configuration: "ran-du.redhat.com"
spec:
 additionalKernelArgs:
 - "rcupdate.rcu_normal_after_boot=0"
 - "efi=runtime"
 - "vfio_pci.enable_sriov=1"
```

```

- "vfio_pci.disable_idle_d3=1"
- "module_blacklist=irdma"

cpu:
 isolated: $isolated
 reserved: $reserved

hugepages:
 defaultHugepagesSize: $defaultHugepagesSize

pages:
 - size: $size
 count: $count
 node: $node

machineConfigPoolSelector:
 pools.operator.machineconfiguration.openshift.io/$mcp: ""

nodeSelector:
 node-role.kubernetes.io/$mcp: ""

numa:
 topologyPolicy: "restricted"

To use the standard (non-realtime) kernel, set enabled to false

realTimeKernel:
 enabled: true

workloadHints:
 # WorkloadHints defines the set of upper level flags for different type of workloads.
 # See https://github.com/openshift/cluster-node-tuning-operator/blob/master/docs/performanceprofile/performance_profile.md#workloadhints
 # for detailed descriptions of each item.
 # The configuration below is set for a low latency, performance mode.
 realTime: true
 highPowerConsumption: false
 perPodPowerManagement: false

```

Table 22.1. PerformanceProfile CR options for single-node OpenShift clusters

PerformanceProfile CR field	Description
<b>metadata.name</b>	<p>Ensure that <b>name</b> matches the following fields set in related GitOps ZTP custom resources (CRs):</p> <ul style="list-style-type: none"> <li>• <b>include=openshift-node-performance-\${PerformanceProfile.metadata.name}</b> in <b>TunedPerformancePatch.yaml</b></li> <li>• <b>name: 50-performance-\${PerformanceProfile.metadata.name}</b> in <b>validatorCRs/informDuValidator.yaml</b></li> </ul>
<b>spec.additionalKernelArgs</b>	" <b>efi=runtime</b> " Configures UEFI secure boot for the cluster host.

PerformanceProfile CR field	Description
<b>spec.cpu.isolated</b>	<p>Set the isolated CPUs. Ensure all of the Hyper-Threading pairs match.</p>  <p><b>IMPORTANT</b></p> <p>The reserved and isolated CPU pools must not overlap and together must span all available cores. CPU cores that are not accounted for cause an undefined behaviour in the system.</p>
<b>spec.cpu.reserved</b>	<p>Set the reserved CPUs. When workload partitioning is enabled, system processes, kernel threads, and system container threads are restricted to these CPUs. All CPUs that are not isolated should be reserved.</p>
<b>spec.hugepages.pages</b>	<ul style="list-style-type: none"> <li>Set the number of huge pages (<b>count</b>)</li> <li>Set the huge pages size (<b>size</b>).</li> <li>Set <b>node</b> to the NUMA node where the <b>hugepages</b> are allocated (<b>node</b>)</li> </ul>
<b>spec.realTimeKernel</b>	<p>Set <b>enabled</b> to <b>true</b> to use the realtime kernel.</p>
<b>spec.workloadHints</b>	<p>Use <b>workloadHints</b> to define the set of top level flags for different type of workloads. The example configuration configures the cluster for low latency and high performance.</p>

## Additional resources

- Recommended single-node OpenShift cluster configuration for vDU application workloads → [Workload partitioning](#)

# CHAPTER 23. USING THE NODE OBSERVABILITY OPERATOR

The Node Observability Operator collects and stores CRI-O and Kubelet profiling or metrics from scripts of compute nodes.

With the Node Observability Operator, you can query the profiling data, enabling analysis of performance trends in CRI-O and Kubelet. It supports debugging performance-related issues and executing embedded scripts for network metrics by using the **run** field in the custom resource definition. To enable CRI-O and Kubelet profiling or scripting, you can configure the **type** field in the custom resource definition.



## IMPORTANT

The Node Observability Operator 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](#).

## 23.1. WORKFLOW OF THE NODE OBSERVABILITY OPERATOR

The following workflow outlines on how to query the profiling data using the Node Observability Operator:

1. Install the Node Observability Operator in the OpenShift Container Platform cluster.
2. Create a NodeObservability custom resource to enable the CRI-O profiling on the worker nodes of your choice.
3. Run the profiling query to generate the profiling data.

## 23.2. INSTALLING THE NODE OBSERVABILITY OPERATOR

The Node Observability Operator is not installed in OpenShift Container Platform by default. You can install the Node Observability Operator by using the OpenShift Container Platform CLI or the web console.

### 23.2.1. Installing the Node Observability Operator using the CLI

You can install the Node Observability Operator by using the OpenShift CLI (oc).

#### Prerequisites

- You have installed the OpenShift CLI (oc).
- You have access to the cluster with **cluster-admin** privileges.

#### Procedure

1. Confirm that the Node Observability Operator is available by running the following command:

```
$ oc get packagemanifests -n openshift-marketplace node-observability-operator
```

### Example output

NAME	CATALOG	AGE
node-observability-operator	Red Hat Operators	9h

2. Create the **node-observability-operator** namespace by running the following command:

```
$ oc new-project node-observability-operator
```

3. Create an **OperatorGroup** object YAML file:

```
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
 name: node-observability-operator
 namespace: node-observability-operator
spec:
 targetNamespaces: []
EOF
```

4. Create a **Subscription** object YAML file to subscribe a namespace to an Operator:

```
cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
 name: node-observability-operator
 namespace: node-observability-operator
spec:
 channel: alpha
 name: node-observability-operator
 source: redhat-operators
 sourceNamespace: openshift-marketplace
EOF
```

## Verification

1. View the install plan name by running the following command:

```
$ oc -n node-observability-operator get sub node-observability-operator -o yaml | yq '.status.installplan.name'
```

### Example output

```
install-dt54w
```

2. Verify the install plan status by running the following command:

```
$ oc -n node-observability-operator get ip <install_plan_name> -o yaml | yq '.status.phase'
```

**<install\_plan\_name>** is the install plan name that you obtained from the output of the previous command.

### Example output

COMPLETE

3. Verify that the Node Observability Operator is up and running:

```
$ oc get deploy -n node-observability-operator
```

### Example output

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
node-observability-operator-controller-manager	1/1	1	1	40h

## 23.2.2. Installing the Node Observability Operator using the web console

You can install the Node Observability Operator from the OpenShift Container Platform web console.

### Prerequisites

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

### Procedure

1. Log in to the OpenShift Container Platform web console.
2. In the Administrator's navigation panel, expand **Operators** → **OperatorHub**.
3. In the **All items** field, enter **Node Observability Operator** and select the **Node Observability Operator** tile.
4. Click **Install**.
5. On the **Install Operator** page, configure the following settings:
  - a. In the **Update channel** area, click **alpha**.
  - b. In the **Installation mode** area, click **A specific namespace on the cluster**
  - c. From the **Installed Namespace** list, select **node-observability-operator** from the list.
  - d. In the **Update approval** area, select **Automatic**.
  - e. Click **Install**.

### Verification

1. In the Administrator's navigation panel, expand **Operators** → **Installed Operators**.
2. Verify that the Node Observability Operator is listed in the Operators list.

## 23.3. REQUESTING CRI-O AND KUBELET PROFILING DATA USING THE NODE OBSERVABILITY OPERATOR

Creating a Node Observability custom resource to collect CRI-O and Kubelet profiling data.

### 23.3.1. Creating the Node Observability custom resource

You must create and run the **NodeObservability** custom resource (CR) before you run the profiling query. When you run the **NodeObservability** CR, it creates the necessary machine config and machine config pool CRs to enable the CRI-O profiling on the worker nodes matching the **nodeSelector**.



#### IMPORTANT

If CRI-O profiling is not enabled on the worker nodes, the **NodeObservabilityMachineConfig** resource gets created. Worker nodes matching the **nodeSelector** specified in **NodeObservability** CR restarts. This might take 10 or more minutes to complete.



#### NOTE

Kubelet profiling is enabled by default.

The CRI-O unix socket of the node is mounted on the agent pod, which allows the agent to communicate with CRI-O to run the pprof request. Similarly, the **kubelet-serving-ca** certificate chain is mounted on the agent pod, which allows secure communication between the agent and node's kubelet endpoint.

#### Prerequisites

- You have installed the Node Observability Operator.
- You have installed the OpenShift CLI (oc).
- You have access to the cluster with **cluster-admin** privileges.

#### Procedure

1. Log in to the OpenShift Container Platform CLI by running the following command:

```
$ oc login -u kubeadmin https://<HOSTNAME>:6443
```

2. Switch back to the **node-observability-operator** namespace by running the following command:

```
$ oc project node-observability-operator
```

3. Create a CR file named **nodeobservability.yaml** that contains the following text:

```
apiVersion: nodeobservability.olm.openshift.io/v1alpha2
kind: NodeObservability
metadata:
 name: cluster ①
spec:
```

```
nodeSelector:
 kubernetes.io/hostname: <node_hostname> 2
 type: crio-kubelet
```

- 1** You must specify the name as **cluster** because there should be only one **NodeObservability** CR per cluster.
- 2** Specify the nodes on which the Node Observability agent must be deployed.

#### 4. Run the **NodeObservability** CR:

```
oc apply -f nodeobservability.yaml
```

#### Example output

```
nodeobservability.olm.openshift.io/cluster created
```

#### 5. Review the status of the **NodeObservability** CR by running the following command:

```
$ oc get nob/cluster -o yaml | jq '.status.conditions'
```

#### Example output

```
conditions:
 conditions:
 - lastTransitionTime: "2022-07-05T07:33:54Z"
 message: 'DaemonSet node-observability-ds ready: true NodeObservabilityMachineConfig
 ready: true'
 reason: Ready
 status: "True"
 type: Ready
```

**NodeObservability** CR run is completed when the reason is **Ready** and the status is **True**.

### 23.3.2. Running the profiling query

To run the profiling query, you must create a **NodeObservabilityRun** resource. The profiling query is a blocking operation that fetches CRI-O and Kubelet profiling data for a duration of 30 seconds. After the profiling query is complete, you must retrieve the profiling data inside the container file system **/run/node-observability** directory. The lifetime of data is bound to the agent pod through the **emptyDir** volume, so you can access the profiling data while the agent pod is in the **running** status.



#### IMPORTANT

You can request only one profiling query at any point of time.

#### Prerequisites

- You have installed the Node Observability Operator.
- You have created the **NodeObservability** custom resource (CR).

- You have access to the cluster with **cluster-admin** privileges.

## Procedure

1. Create a **NodeObservabilityRun** resource file named **nodeobservabilityrun.yaml** that contains the following text:

```
apiVersion: nodeobservability.olm.openshift.io/v1alpha2
kind: NodeObservabilityRun
metadata:
 name: nodeobservabilityrun
spec:
 nodeObservabilityRef:
 name: cluster
```

2. Trigger the profiling query by running the **NodeObservabilityRun** resource:

```
$ oc apply -f nodeobservabilityrun.yaml
```

3. Review the status of the **NodeObservabilityRun** by running the following command:

```
$ oc get nodeobservabilityrun nodeobservabilityrun -o yaml | jq '.status.conditions'
```

## Example output

```
conditions:
- lastTransitionTime: "2022-07-07T14:57:34Z"
 message: Ready to start profiling
 reason: Ready
 status: "True"
 type: Ready
- lastTransitionTime: "2022-07-07T14:58:10Z"
 message: Profiling query done
 reason: Finished
 status: "True"
 type: Finished
```

The profiling query is complete once the status is **True** and type is **Finished**.

4. Retrieve the profiling data from the container's **/run/node-observability** path by running the following bash script:

```
for a in $(oc get nodeobservabilityrun nodeobservabilityrun -o yaml | jq ".status.agents[].name"); do
 echo "agent ${a}"
 mkdir -p "/tmp/${a}"
 for p in $(oc exec "${a}" -c node-observability-agent -- bash -c "ls /run/node-observability/* .pprof"); do
 f=$(basename ${p})
 echo "copying ${f} to /tmp/${a}/${f}"
 oc exec "${a}" -c node-observability-agent -- cat "${p}" > "/tmp/${a}/${f}"
 done
done
```

## 23.4. NODE OBSERVABILITY OPERATOR SCRIPTING

Scripting allows you to run pre-configured bash scripts, using the current Node Observability Operator and Node Observability Agent.

These scripts monitor key metrics like CPU load, memory pressure, and worker node issues. They also collect sar reports and custom performance metrics.

### 23.4.1. Creating the Node Observability custom resource for scripting

You must create and run the **NodeObservability** custom resource (CR) before you run the scripting. When you run the **NodeObservability** CR, it enables the agent in scripting mode on the compute nodes matching the **nodeSelector** label.

#### Prerequisites

- You have installed the Node Observability Operator.
- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster with **cluster-admin** privileges.

#### Procedure

1. Log in to the OpenShift Container Platform cluster by running the following command:

```
$ oc login -u kubeadmin https://<host_name>:6443
```

2. Switch to the **node-observability-operator** namespace by running the following command:

```
$ oc project node-observability-operator
```

3. Create a file named **nodeobservability.yaml** that contains the following content:

```
apiVersion: nodeobservability.olm.openshift.io/v1alpha2
kind: NodeObservability
metadata:
 name: cluster 1
spec:
 nodeSelector:
 kubernetes.io/hostname: <node_hostname> 2
 type: scripting 3
```

**1** You must specify the name as **cluster** because there should be only one **NodeObservability** CR per cluster.

**2** Specify the nodes on which the Node Observability agent must be deployed.

**3** To deploy the agent in scripting mode, you must set the type to **scripting**.

4. Create the **NodeObservability** CR by running the following command:

```
$ oc apply -f nodeobservability.yaml
```

## Example output

```
nodeobservability.olm.openshift.io/cluster created
```

- Review the status of the **NodeObservability** CR by running the following command:

```
$ oc get nob/cluster -o yaml | yq '.status.conditions'
```

## Example output

```
conditions:
 conditions:
 - lastTransitionTime: "2022-07-05T07:33:54Z"
 message: 'DaemonSet node-observability-ds ready: true NodeObservabilityScripting
 ready: true'
 reason: Ready
 status: "True"
 type: Ready
```

The **NodeObservability** CR run is completed when the **reason** is **Ready** and **status** is "**True**".

### 23.4.2. Configuring Node Observability Operator scripting

#### Prerequisites

- You have installed the Node Observability Operator.
- You have created the **NodeObservability** custom resource (CR).
- You have access to the cluster with **cluster-admin** privileges.

#### Procedure

- Create a file named **nodeobservabilityrun-script.yaml** that contains the following content:

```
apiVersion: nodeobservability.olm.openshift.io/v1alpha2
kind: NodeObservabilityRun
metadata:
 name: nodeobservabilityrun-script
 namespace: node-observability-operator
spec:
 nodeObservabilityRef:
 name: cluster
 type: scripting
```

#### IMPORTANT

You can request only the following scripts:

- metrics.sh**
- network-metrics.sh** (uses **monitor.sh**)

- Trigger the scripting by creating the **NodeObservabilityRun** resource with the following command:

```
$ oc apply -f nodeobservabilityrun-script.yaml
```

- Review the status of the **NodeObservabilityRun** scripting by running the following command:

```
$ oc get nodeobservabilityrun nodeobservabilityrun-script -o yaml | jq '.status.conditions'
```

### Example output

```
Status:
Agents:
Ip: 10.128.2.252
Name: node-observability-agent-n2fpm
Port: 8443
Ip: 10.131.0.186
Name: node-observability-agent-wcc8p
Port: 8443
Conditions:
Conditions:
Last Transition Time: 2023-12-19T15:10:51Z
Message: Ready to start profiling
Reason: Ready
Status: True
Type: Ready
Last Transition Time: 2023-12-19T15:11:01Z
Message: Profiling query done
Reason: Finished
Status: True
Type: Finished
Finished Timestamp: 2023-12-19T15:11:01Z
Start Timestamp: 2023-12-19T15:10:51Z
```

The scripting is complete once **Status** is **True** and **Type** is **Finished**.

- Retrieve the scripting data from the root path of the container by running the following bash script:

```
#!/bin/bash

RUN=$(oc get nodeobservabilityrun --no-headers | awk '{print $1}')

for a in $(oc get nodeobservabilityruns.nodeobservability.olm.openshift.io/${RUN} -o json | jq .status.agents[].name); do
 echo "agent ${a}"
 agent=$(echo ${a} | tr -d "\\"\\\"")
 base_dir=$(oc exec "${agent}" -c node-observability-agent -- bash -c "ls -t | grep node-observability-agent" | head -1)
 echo "${base_dir}"
 mkdir -p "/tmp/${agent}"
 for p in $(oc exec "${agent}" -c node-observability-agent -- bash -c "ls ${base_dir}"); do
 f="/${base_dir}/${p}"
 echo "copying ${f} to /tmp/${agent}/${p}"
 done
done
```

```
oc exec "${agent}" -c node-observability-agent -- cat ${f} > "/tmp/${agent}/${p}"
done
done
```

## 23.5. ADDITIONAL RESOURCES

For more information on how to collect worker metrics, see [Red Hat Knowledgebase article](#).