



OpenShift Container Platform 4.11

Scalability and performance

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

OpenShift Container Platform 4.11 Scalability and performance

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

Legal Notice

Copyright © 2022 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

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

Table of Contents

CHAPTER 1. RECOMMENDED PRACTICES FOR INSTALLING LARGE CLUSTERS	10
1.1. RECOMMENDED PRACTICES FOR INSTALLING LARGE SCALE CLUSTERS	10
CHAPTER 2. RECOMMENDED HOST PRACTICES	11
2.1. RECOMMENDED NODE HOST PRACTICES	11
2.2. CREATING A KUBELETCONFIG CRD TO EDIT KUBELET PARAMETERS	12
2.3. MODIFYING THE NUMBER OF UNAVAILABLE WORKER NODES	16
2.4. CONTROL PLANE NODE SIZING	16
2.4.1. Increasing the flavor size of the Amazon Web Services (AWS) master instances	18
2.5. RECOMMENDED ETCD PRACTICES	19
2.6. DEFRAGMENTING ETCD DATA	21
2.6.1. Automatic defragmentation	21
2.6.2. Manual defragmentation	22
2.7. OPENSIFT CONTAINER PLATFORM INFRASTRUCTURE COMPONENTS	25
2.8. MOVING THE MONITORING SOLUTION	25
2.9. MOVING THE DEFAULT REGISTRY	26
2.10. MOVING THE ROUTER	28
2.11. INFRASTRUCTURE NODE SIZING	29
2.12. ADDITIONAL RESOURCES	30
CHAPTER 3. RECOMMENDED HOST PRACTICES FOR IBM Z & LINUXONE ENVIRONMENTS	31
3.1. MANAGING CPU OVERCOMMITMENT	31
3.2. DISABLE TRANSPARENT HUGE PAGES	31
3.3. BOOST NETWORKING PERFORMANCE WITH RECEIVE FLOW STEERING	32
3.3.1. Use the Machine Config Operator (MCO) to activate RFS	32
3.4. CHOOSE YOUR NETWORKING SETUP	33
3.5. ENSURE HIGH DISK PERFORMANCE WITH HYPERPAV ON Z/VM	33
3.5.1. Use the Machine Config Operator (MCO) to activate HyperPAV aliases in nodes using z/VM full-pack minidisks	34
3.6. RHEL KVM ON IBM Z HOST RECOMMENDATIONS	35
3.6.1. Use multiple queues for your VirtIO network interfaces	35
3.6.2. Use I/O threads for your virtual block devices	35
3.6.3. Avoid virtual SCSI devices	36
3.6.4. Configure guest caching for disk	36
3.6.5. Exclude the memory balloon device	37
3.6.6. Tune the CPU migration algorithm of the host scheduler	37
3.6.7. Disable the cpuset cgroup controller	37
3.6.8. Tune the polling period for idle virtual CPUs	38
CHAPTER 4. RECOMMENDED CLUSTER SCALING PRACTICES	39
4.1. RECOMMENDED PRACTICES FOR SCALING THE CLUSTER	39
4.2. MODIFYING A MACHINE SET	39
4.3. ABOUT MACHINE HEALTH CHECKS	41
4.3.1. Limitations when deploying machine health checks	41
4.4. SAMPLE MACHINEHEALTHCHECK RESOURCE	42
4.4.1. Short-circuiting machine health check remediation	43
4.4.1.1. Setting maxUnhealthy by using an absolute value	43
4.4.1.2. Setting maxUnhealthy by using percentages	43
4.5. CREATING A MACHINEHEALTHCHECK RESOURCE	44
CHAPTER 5. USING THE NODE TUNING OPERATOR	45
5.1. ABOUT THE NODE TUNING OPERATOR	45

5.2. ACCESSING AN EXAMPLE NODE TUNING OPERATOR SPECIFICATION	45
5.3. DEFAULT PROFILES SET ON A CLUSTER	46
5.4. VERIFYING THAT THE TUNED PROFILES ARE APPLIED	46
5.5. CUSTOM TUNING SPECIFICATION	47
5.6. CUSTOM TUNING EXAMPLES	52
5.7. SUPPORTED TUNED DAEMON PLUG-INS	53
CHAPTER 6. USING CPU MANAGER	55
6.1. SETTING UP CPU MANAGER	55
CHAPTER 7. USING TOPOLOGY MANAGER	60
7.1. TOPOLOGY MANAGER POLICIES	60
7.2. SETTING UP TOPOLOGY MANAGER	61
7.3. POD INTERACTIONS WITH TOPOLOGY MANAGER POLICIES	61
CHAPTER 8. SCHEDULING NUMA-AWARE WORKLOADS	63
8.1. ABOUT NUMA-AWARE SCHEDULING	63
8.2. INSTALLING THE NUMA RESOURCES OPERATOR	64
8.2.1. Installing the NUMA Resources Operator using the CLI	64
8.2.2. Installing the NUMA Resources Operator using the web console	66
8.3. CREATING THE NUMARESOURCESOPERATOR CUSTOM RESOURCE	66
8.4. DEPLOYING THE NUMA-AWARE SECONDARY POD SCHEDULER	68
8.5. SCHEDULING WORKLOADS WITH THE NUMA-AWARE SCHEDULER	70
8.6. TROUBLESHOOTING NUMA-AWARE SCHEDULING	73
8.6.1. Checking the NUMA-aware scheduler logs	76
8.6.2. Troubleshooting the resource topology exporter	79
8.6.3. Correcting a missing resource topology exporter config map	80
CHAPTER 9. SCALING THE CLUSTER MONITORING OPERATOR	83
9.1. PROMETHEUS DATABASE STORAGE REQUIREMENTS	83
9.2. CONFIGURING CLUSTER MONITORING	84
CHAPTER 10. PLANNING YOUR ENVIRONMENT ACCORDING TO OBJECT MAXIMUMS	86
10.1. OPENSIFT CONTAINER PLATFORM TESTED CLUSTER MAXIMUMS FOR MAJOR RELEASES	86
10.2. OPENSIFT CONTAINER PLATFORM ENVIRONMENT AND CONFIGURATION ON WHICH THE CLUSTER MAXIMUMS ARE TESTED	88
10.2.1. AWS cloud platform	88
10.2.2. IBM Power platform	88
10.2.3. IBM Z platform	89
10.3. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO TESTED CLUSTER MAXIMUMS	90
10.4. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO APPLICATION REQUIREMENTS	90
CHAPTER 11. OPTIMIZING STORAGE	94
11.1. AVAILABLE PERSISTENT STORAGE OPTIONS	94
11.2. RECOMMENDED CONFIGURABLE STORAGE TECHNOLOGY	95
11.2.1. Specific application storage recommendations	95
11.2.1.1. Registry	96
11.2.1.2. Scaled registry	96
11.2.1.3. Metrics	96
11.2.1.4. Logging	97
11.2.1.5. Applications	97
11.2.2. Other specific application storage recommendations	97
11.3. DATA STORAGE MANAGEMENT	97
CHAPTER 12. OPTIMIZING ROUTING	99

12.1. BASELINE INGRESS CONTROLLER (ROUTER) PERFORMANCE	99
12.2. INGRESS CONTROLLER (ROUTER) PERFORMANCE OPTIMIZATIONS	100
12.2.1. Configuring Ingress Controller liveness, readiness, and startup probes	100
CHAPTER 13. OPTIMIZING NETWORKING	102
13.1. OPTIMIZING THE MTU FOR YOUR NETWORK	102
13.2. RECOMMENDED PRACTICES FOR INSTALLING LARGE SCALE CLUSTERS	103
13.3. IMPACT OF IPSEC	103
13.4. ADDITIONAL RESOURCES	103
CHAPTER 14. MANAGING BARE METAL HOSTS	104
14.1. ABOUT BARE METAL HOSTS AND NODES	104
14.2. MAINTAINING BARE METAL HOSTS	104
14.2.1. Adding a bare metal host to the cluster using the web console	104
14.2.2. Adding a bare metal host to the cluster using YAML in the web console	105
14.2.3. Automatically scaling machines to the number of available bare metal hosts	106
CHAPTER 15. WHAT HUGE PAGES DO AND HOW THEY ARE CONSUMED BY APPLICATIONS	108
15.1. WHAT HUGE PAGES DO	108
15.2. HOW HUGE PAGES ARE CONSUMED BY APPS	108
15.3. CONSUMING HUGE PAGES RESOURCES USING THE DOWNWARD API	109
15.4. CONFIGURING HUGE PAGES	111
15.4.1. At boot time	111
15.5. DISABLING TRANSPARENT HUGE PAGES	113
CHAPTER 16. LOW LATENCY TUNING	115
16.1. UNDERSTANDING LOW LATENCY	115
16.1.1. About hyperthreading for low latency and real-time applications	116
16.2. PROVISIONING REAL-TIME AND LOW LATENCY WORKLOADS	116
16.2.1. Known limitations for real-time	117
16.2.2. Provisioning a worker with real-time capabilities	117
16.2.3. Verifying the real-time kernel installation	119
16.2.4. Creating a workload that works in real-time	119
16.2.5. Creating a pod with a QoS class of Guaranteed	119
16.2.6. Optional: Disabling CPU load balancing for DPDK	120
16.2.7. Assigning a proper node selector	121
16.2.8. Scheduling a workload onto a worker with real-time capabilities	121
16.2.9. Reducing power consumption by taking CPUs offline	122
16.2.10. Managing device interrupt processing for guaranteed pod isolated CPUs	122
16.2.10.1. Disabling CPU CFS quota	123
16.2.10.2. Disabling global device interrupts handling in Node Tuning Operator	123
16.2.10.3. Disabling interrupt processing for individual pods	123
16.2.11. Upgrading the performance profile to use device interrupt processing	124
16.2.11.1. Supported API Versions	124
16.2.11.1.1. Upgrading Node Tuning Operator API from v1alpha1 to v1	124
16.2.11.1.2. Upgrading Node Tuning Operator API from v1alpha1 or v1 to v2	124
16.3. TUNING NODES FOR LOW LATENCY WITH THE PERFORMANCE PROFILE	124
16.3.1. Configuring huge pages	125
16.3.2. Allocating multiple huge page sizes	127
16.3.3. Configuring a node for IRQ dynamic load balancing	127
16.3.4. Configuring hyperthreading for a cluster	130
16.3.4.1. Disabling hyperthreading for low latency applications	131
16.3.5. Understanding workload hints	132
16.3.6. Configuring workload hints manually	133

16.3.7. Restricting CPUs for infra and application containers	133
16.4. REDUCING NIC QUEUES USING THE NODE TUNING OPERATOR	136
16.4.1. Adjusting the NIC queues with the performance profile	136
16.4.2. Verifying the queue status	139
16.4.3. Logging associated with adjusting NIC queues	143
16.5. PERFORMING END-TO-END TESTS FOR PLATFORM VERIFICATION	143
16.5.1. Prerequisites	144
16.5.2. Dry run	144
16.5.3. Disconnected mode	145
16.5.3.1. Mirroring the images to a custom registry accessible from the cluster	145
16.5.3.2. Instruct the tests to consume those images from a custom registry	145
16.5.3.3. Mirroring to the cluster internal registry	145
16.5.3.4. Mirroring a different set of images	147
16.5.4. Running in a single-node cluster	147
Required parameters	148
16.5.5. Impact of tests on the cluster	148
16.5.5.1. SCTP	148
16.5.5.2. XT_U32	148
16.5.5.3. SR-IOV	148
16.5.5.4. PTP	148
16.5.5.5. Performance	148
16.5.5.6. DPDK	149
16.5.5.7. Container-mount-namespace	149
16.5.5.8. Cleaning up	149
16.5.6. Override test image parameters	149
16.5.6.1. Ginkgo parameters	149
16.5.6.2. Available features	150
16.5.7. Discovery mode	150
16.5.7.1. Required environment configuration prerequisites	150
16.5.7.2. Limiting the nodes used during tests	151
16.5.7.3. Using a single performance profile	152
16.5.7.4. Disabling the performance profile cleanup	152
16.5.8. Running the latency tests	153
16.5.8.1. Running hwlatdetect	154
16.5.8.1.1. Capturing the results	158
16.5.8.2. Running cyclicttest	159
16.5.8.2.1. Capturing the results	162
16.5.8.3. Running oslat	163
16.5.9. Troubleshooting	166
16.5.10. Test reports	166
16.5.10.1. JUnit test output	166
16.5.10.2. Test failure report	166
16.5.10.3. A note on podman	166
16.5.10.4. Running on OpenShift Container Platform 4.4	166
16.5.10.5. Using a single performance profile	167
16.6. DEBUGGING LOW LATENCY CNF TUNING STATUS	167
16.6.1. Machine config pools	169
16.7. COLLECTING LOW LATENCY TUNING DEBUGGING DATA FOR RED HAT SUPPORT	170
16.7.1. About the must-gather tool	170
16.7.2. About collecting low latency tuning data	170
16.7.3. Gathering data about specific features	171

CHAPTER 17. IMPROVING CLUSTER STABILITY IN HIGH LATENCY ENVIRONMENTS USING WORKER

LATENCY PROFILES	173
17.1. UNDERSTANDING WORKER LATENCY PROFILES	173
17.2. USING WORKER LATENCY PROFILES	175
CHAPTER 18. TOPOLOGY AWARE LIFECYCLE MANAGER FOR CLUSTER UPDATES	179
18.1. ABOUT THE TOPOLOGY AWARE LIFECYCLE MANAGER CONFIGURATION	179
18.2. ABOUT MANAGED POLICIES USED WITH TOPOLOGY AWARE LIFECYCLE MANAGER	179
18.3. INSTALLING THE TOPOLOGY AWARE LIFECYCLE MANAGER BY USING THE WEB CONSOLE	180
18.4. INSTALLING THE TOPOLOGY AWARE LIFECYCLE MANAGER BY USING THE CLI	181
18.5. ABOUT THE CLUSTERGROUPUPGRADE CR	182
18.5.1. The UpgradeNotStarted state	182
18.5.2. The UpgradeCannotStart state	184
18.5.3. The UpgradeNotCompleted state	184
18.5.4. The UpgradeTimedOut state	186
18.5.5. The UpgradeCompleted state	186
18.5.6. Blocking ClusterGroupUpgrade CRs	187
18.6. UPDATE POLICIES ON MANAGED CLUSTERS	194
18.6.1. Applying update policies to managed clusters	194
18.7. CREATING A BACKUP OF CLUSTER RESOURCES BEFORE UPGRADE	201
18.7.1. Creating a ClusterGroupUpgrade CR with backup	201
18.7.2. Recovering a cluster after a failed upgrade	203
18.8. USING THE CONTAINER IMAGE PRE-CACHE FEATURE	206
18.8.1. Creating a ClusterGroupUpgrade CR with pre-caching	206
18.9. TROUBLESHOOTING THE TOPOLOGY AWARE LIFECYCLE MANAGER	209
18.9.1. General troubleshooting	209
18.9.2. Cannot modify the ClusterUpgradeGroup CR	210
18.9.3. Managed policies	210
Checking managed policies on the system	210
Checking remediationAction mode	211
Checking policy compliance state	211
18.9.4. Clusters	212
Checking if managed clusters are present	212
Checking if managed clusters are available	212
Checking clusterSelector	213
Checking if canary clusters are present	213
Checking the pre-caching status on spoke clusters	214
18.9.5. Remediation Strategy	214
Checking if remediationStrategy is present in the ClusterGroupUpgrade CR	214
Checking if maxConcurrency is specified in the ClusterGroupUpgrade CR	215
18.9.6. Topology Aware Lifecycle Manager	215
Checking condition message and status in the ClusterGroupUpgrade CR	215
Checking corresponding copied policies	215
Checking if status.remediationPlan was computed	216
Errors in the TALM manager container	216
CHAPTER 19. CREATING A PERFORMANCE PROFILE	217
19.1. ABOUT THE PERFORMANCE PROFILE CREATOR	217
19.1.1. Gathering data about your cluster using must-gather	217
19.1.2. Running the Performance Profile Creator using podman	218
19.1.2.1. How to run podman to create a performance profile	221
19.1.3. Running the Performance Profile Creator wrapper script	222
19.1.4. Performance Profile Creator arguments	226
19.2. REFERENCE PERFORMANCE PROFILES	229

19.2.1. A performance profile template for clusters that use OVS-DPDK on OpenStack	229
19.3. ADDITIONAL RESOURCES	230
CHAPTER 20. DEPLOYING DISTRIBUTED UNITS MANUALLY ON SINGLE-NODE OPENSIFT	231
20.1. CONFIGURING THE DISTRIBUTED UNITS (DUS)	231
20.1.1. Enabling workload partitioning	231
20.1.2. Configuring the container mount namespace	233
20.1.3. Enabling Stream Control Transmission Protocol (SCTP)	234
20.1.4. Creating OperatorGroups for Operators	235
20.1.5. Subscribing to the Operators	237
20.1.6. Configuring logging locally and forwarding	238
20.1.7. Configuring the Node Tuning Operator	239
20.1.8. Configuring Precision Time Protocol (PTP)	240
20.1.9. Disabling Network Time Protocol (NTP)	242
20.1.10. Configuring single root I/O virtualization (SR-IOV)	243
20.1.11. Disabling the console Operator	245
20.2. APPLYING THE DISTRIBUTED UNIT (DU) CONFIGURATION TO A SINGLE-NODE OPENSIFT CLUSTER	245
20.2.1. Applying the extra installation manifests	245
20.2.2. Applying the post-install configuration custom resources (CRs)	245
CHAPTER 21. VALIDATING CLUSTER TUNING FOR VDU APPLICATION WORKLOADS	246
21.1. RECOMMENDED BIOS CONFIGURATION FOR VDU CLUSTER HOSTS	246
21.2. RECOMMENDED CLUSTER CONFIGURATIONS TO RUN VDU APPLICATIONS	247
21.2.1. Recommended cluster MachineConfig CRs	248
21.2.2. Recommended cluster Operators	248
21.2.3. Recommended cluster kernel configuration	248
21.2.4. Checking the realtime kernel version	249
21.3. CHECKING THAT THE RECOMMENDED CLUSTER CONFIGURATIONS ARE APPLIED	250
CHAPTER 22. WORKLOAD PARTITIONING ON SINGLE-NODE OPENSIFT	261
22.1. ENABLING WORKLOAD PARTITIONING	261
CHAPTER 23. DEPLOYING DISTRIBUTED UNITS AT SCALE IN A DISCONNECTED ENVIRONMENT	264
23.1. PROVISIONING EDGE SITES AT SCALE	264
23.2. ABOUT ZTP AND DISTRIBUTED UNITS ON OPENSIFT CLUSTERS	265
23.3. THE GITOPS APPROACH	265
23.4. ZERO TOUCH PROVISIONING BUILDING BLOCKS	265
23.5. HOW TO PLAN YOUR RAN POLICIES	266
23.6. LOW LATENCY FOR DISTRIBUTED UNITS (DUS)	267
23.7. PREPARING THE DISCONNECTED ENVIRONMENT	268
23.7.1. Adding RHCOS ISO and RootFS images to the disconnected mirror host	268
23.8. INSTALLING RED HAT ADVANCED CLUSTER MANAGEMENT IN A DISCONNECTED ENVIRONMENT	270
23.9. ENABLING ASSISTED INSTALLER SERVICE ON BARE METAL	270
23.10. ZTP CUSTOM RESOURCES	272
23.11. POLICYGENTEMPLATE CRS FOR RAN DEPLOYMENTS	274
23.12. ABOUT THE POLICYGENTEMPLATE	275
23.13. BEST PRACTICES WHEN CUSTOMIZING POLICYGENTEMPLATE CRS	278
23.14. CREATING THE POLICYGENTEMPLATE CR	278
23.15. CONFIGURING POLICY COMPLIANCE EVALUATION TIMEOUTS FOR POLICYGENTEMPLATE CRS	280
23.16. CREATING ZTP CUSTOM RESOURCES FOR MULTIPLE MANAGED CLUSTERS	281
23.16.1. Using PolicyGenTemplate CRs to override source CRs content	282

23.16.2. Filtering custom resources using SiteConfig filters	285
23.16.3. Configuring PTP fast events using PolicyGenTemplate CRs	286
23.16.4. Configuring UEFI secure boot for clusters using PolicyGenTemplate CRs	288
23.16.5. Configuring bare-metal event monitoring using PolicyGenTemplate CRs	289
23.17. INSTALLING THE GITOPS ZTP PIPELINE	291
23.17.1. Preparing the ZTP Git repository	291
23.17.2. Preparing the hub cluster for ZTP	292
23.17.3. Deploying additional changes to clusters	293
23.18. ADDING NEW CONTENT TO THE GITOPS ZTP PIPELINE	294
23.19. CUSTOMIZING EXTRA INSTALLATION MANIFESTS IN THE ZTP GITOPS PIPELINE	295
23.20. DEPLOYING A SITE	296
23.21. GITOPS ZTP AND TOPOLOGY AWARE LIFECYCLE MANAGER	300
23.22. MONITORING DEPLOYMENT PROGRESS	302
23.23. INDICATION OF DONE FOR ZTP INSTALLATIONS	303
23.23.1. Creating a validator inform policy	303
23.23.2. Querying the policy compliance status for each cluster	305
23.23.3. Node Tuning Operator	306
23.24. TROUBLESHOOTING GITOPS ZTP	306
23.24.1. Validating the generation of installation CRs	306
23.24.2. Validating the generation of configuration policy CRs	307
23.24.3. Restarting policies reconciliation	309
23.25. SITE CLEANUP	310
23.25.1. Removing obsolete content	310
23.25.2. Tearing down the pipeline	311
23.26. UPGRADING GITOPS ZTP	311
23.26.1. Preparing for the upgrade	312
23.26.2. Labeling the existing clusters	312
23.26.3. Stopping the existing GitOps ZTP applications	313
23.26.4. Topology Aware Lifecycle Manager	313
23.26.5. Required changes to the Git repository	313
23.26.6. Installing the new GitOps ZTP applications	315
23.26.7. Roll out the configuration changes	315
23.27. MANUALLY INSTALL A SINGLE MANAGED CLUSTER	315
23.27.1. Configuring BIOS for distributed unit bare-metal hosts	321
23.27.2. Configuring static IP addresses for managed clusters	322
23.27.3. Automated Discovery image ISO process for provisioning clusters	323
23.27.4. Checking the managed cluster status	324
23.27.5. Configuring a managed cluster for a disconnected environment	325
23.27.6. Configuring IPv6 addresses for a disconnected environment	326
23.28. GENERATING RAN POLICIES	327
23.28.1. Troubleshooting the managed cluster	329
23.29. UPDATING MANAGED POLICIES WITH THE TOPOLOGY AWARE LIFECYCLE MANAGER	330
23.29.1. About the auto-created ClusterGroupUpgrade CR for ZTP	330
23.30. END-TO-END PROCEDURES FOR UPDATING CLUSTERS IN A DISCONNECTED ENVIRONMENT	331
23.30.1. Preparing for the updates	331
23.30.2. Setting up the environment	332
23.30.3. Performing a platform update	333
23.30.4. Performing an Operator update	337
23.30.5. Performing a platform and an Operator update together	342
23.31. REMOVING PERFORMANCE ADDON OPERATOR SUBSCRIPTIONS FROM DEPLOYED CLUSTERS	344

CHAPTER 24. REQUESTING CRI-O AND KUBELET PROFILING DATA BY USING THE NODE OBSERVABILITY

- OPERATOR 347**
 - 24.1. WORKFLOW OF THE NODE OBSERVABILITY OPERATOR 347
 - 24.2. INSTALLING THE NODE OBSERVABILITY OPERATOR 347
 - 24.2.1. Installing the Node Observability Operator using the CLI 347
 - 24.2.2. Installing the Node Observability Operator using the web console 349
 - 24.3. CREATING THE NODE OBSERVABILITY CUSTOM RESOURCE 350
 - 24.4. RUNNING THE PROFILING QUERY 351

CHAPTER 1. RECOMMENDED PRACTICES FOR INSTALLING LARGE CLUSTERS

Apply the following practices when installing large clusters or scaling clusters to larger node counts.

1.1. 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:

```
networking:
  clusterNetwork:
    - cidr: 10.128.0.0/14
      hostPrefix: 23
  machineNetwork:
    - cidr: 10.0.0.0/16
  networkType: OpenShiftSDN
  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. It must be set to **10.128.0.0/12** or **10.128.0.0/10** to get to larger node counts beyond 500 nodes.

CHAPTER 2. RECOMMENDED HOST PRACTICES

This topic provides recommended host practices for OpenShift Container Platform.



IMPORTANT

These guidelines apply to OpenShift Container Platform with software-defined networking (SDN), not Open Virtual Network (OVN).

2.1. RECOMMENDED NODE HOST PRACTICES

The OpenShift Container Platform node configuration file contains important options. For example, two parameters control the maximum number of pods that can be scheduled to a node: **podsPerCore** and **maxPods**.

When both options are in use, the lower of the two values limits the number of pods on a node. Exceeding these values can result in:

- Increased CPU utilization.
- Slow pod scheduling.
- Potential out-of-memory scenarios, depending on the amount of memory in the node.
- Exhausting the pool of IP addresses.
- Resource overcommitting, leading to poor user application performance.



IMPORTANT

In Kubernetes, a pod that is holding a single container actually uses two containers. The second container is used to set up networking prior to the actual container starting. Therefore, a system running 10 pods will actually have 20 containers running.



NOTE

Disk IOPS throttling from the cloud provider might have an impact on CRI-O and kubelet. They might get overloaded when there are large number of I/O intensive pods running on the nodes. It is recommended that you monitor the disk I/O on the nodes and use volumes with sufficient throughput for the workload.

podsPerCore sets the number of pods the node can run based on the number of processor cores on the node. For example, if **podsPerCore** is set to **10** on a node with 4 processor cores, the maximum number of pods allowed on the node will be **40**.

```
kubeletConfig:
  podsPerCore: 10
```

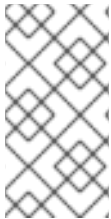
Setting **podsPerCore** to **0** disables this limit. The default is **0**. **podsPerCore** cannot exceed **maxPods**.

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

kubeletConfig:
maxPods: 250

2.2. CREATING A KUBELETCONFIG CRD TO EDIT KUBELET PARAMETERS

The kubelet configuration is currently serialized as an Ignition configuration, so it can be directly edited. However, there is also a new **kubelet-config-controller** added to the Machine Config Controller (MCC). This lets you use a **KubeletConfig** custom resource (CR) to edit the kubelet parameters.



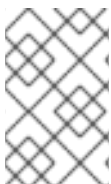
NOTE

As the fields in the **kubeletConfig** object are passed directly to the kubelet from upstream Kubernetes, the kubelet validates those values directly. Invalid values in the **kubeletConfig** object might cause cluster nodes to become unavailable. For valid values, see the [Kubernetes documentation](#).

Consider the following guidance:

- Create one **KubeletConfig** CR for each machine config pool with all the config changes you want for that pool. If you are applying the same content to all of the pools, you need only one **KubeletConfig** CR for all of the pools.
- Edit an existing **KubeletConfig** CR to modify existing settings or add new settings, instead of creating a CR for each change. It is recommended that you create a CR only to modify a different machine config pool, or for changes that are intended to be temporary, so that you can revert the changes.
- As needed, create multiple **KubeletConfig** CRs with a limit of 10 per cluster. For the first **KubeletConfig** CR, the Machine Config Operator (MCO) creates a machine config appended with **kubelet**. With each subsequent CR, the controller creates another **kubelet** machine config with a numeric suffix. For example, if you have a **kubelet** machine config with a **-2** suffix, the next **kubelet** machine config is appended with **-3**.

If you want to delete the machine configs, delete them in reverse order to avoid exceeding the limit. For example, you delete the **kubelet-3** machine config before deleting the **kubelet-2** machine config.



NOTE

If you have a machine config with a **kubelet-9** suffix, and you create another **KubeletConfig** CR, a new machine config is not created, even if there are fewer than 10 **kubelet** machine configs.

Example KubeletConfig CR

```
$ oc get kubeletconfig
```

NAME	AGE
set-max-pods	15m

Example showing a KubeletConfig machine config

■


```
$ oc get mc | grep kubelet
```

```
...
99-worker-generated-kubelet-1      b5c5119de007945b6fe6fb215db3b8e2ceb12511  3.2.0
26m
...
```

The following procedure is an example to show how to configure the maximum number of pods per node on the worker nodes.

Prerequisites

1. Obtain the label associated with the static **MachineConfigPool** CR for the type of node you want to configure. Perform one of the following steps:
 - a. View the machine config pool:

```
$ oc describe machineconfigpool <name>
```

For example:

```
$ oc describe machineconfigpool worker
```

Example output

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: 2019-02-08T14:52:39Z
  generation: 1
  labels:
    custom-kubelet: set-max-pods 1
```

1 If a label has been added it appears under **labels**.

- b. If the label is not present, add a key/value pair:

```
$ oc label machineconfigpool worker custom-kubelet=set-max-pods
```

Procedure

1. View the available machine configuration objects that you can select:

```
$ oc get machineconfig
```

By default, the two kubelet-related configs are **01-master-kubelet** and **01-worker-kubelet**.

2. Check the current value for the maximum pods per node:

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

For example:

```
$ oc describe node ci-ln-5grqprb-f76d1-ncnqq-worker-a-mdv94
```

Look for **value: pods: <value>** in the **Allocatable** stanza:

Example output

```
Allocatable:
attachable-volumes-aws-ebs: 25
cpu:                        3500m
hugepages-1Gi:             0
hugepages-2Mi:             0
memory:                    15341844Ki
pods:                      250
```

- Set the maximum pods per node on the worker nodes by creating a custom resource file that contains the kubelet configuration:

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

❶

Enter the label from the machine config pool.

❷

Add the kubelet configuration. In this example, use **maxPods** to set the maximum pods per node.



NOTE

The rate at which the kubelet talks to the API server depends on queries per second (QPS) and burst values. The default values, **50** for **kubeAPIQPS** and **100** for **kubeAPIBurst**, are sufficient if there are limited pods running on each node. It is recommended to update the kubelet QPS and burst rates if there are enough CPU and memory resources on the node.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: set-max-pods
  kubeletConfig:
    maxPods: <pod_count>
    kubeAPIBurst: <burst_rate>
    kubeAPIQPS: <QPS>
```

- a. Update the machine config pool for workers with the label:

```
$ oc label machineconfigpool worker custom-kubelet=set-max-pods
```

- b. Create the **KubeletConfig** object:

```
$ oc create -f change-maxPods-cr.yaml
```

- c. Verify that the **KubeletConfig** object is created:

```
$ oc get kubeletconfig
```

Example output

```
NAME          AGE
set-max-pods  15m
```

Depending on the number of worker nodes in the cluster, wait for the worker nodes to be rebooted one by one. For a cluster with 3 worker nodes, this could take about 10 to 15 minutes.

4. Verify that the changes are applied to the node:

- a. Check on a worker node that the **maxPods** value changed:

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

- b. Locate the **Allocatable** stanza:

```
...
Allocatable:
attachable-volumes-gce-pd: 127
cpu:                        3500m
ephemeral-storage:         123201474766
hugepages-1Gi:             0
hugepages-2Mi:             0
memory:                    14225400Ki
pods:                      500 1
...
```

1 In this example, the **pods** parameter should report the value you set in the **KubeletConfig** object.

5. Verify the change in the **KubeletConfig** object:

```
$ oc get kubeletconfigs set-max-pods -o yaml
```

This should show a status of **True** and **type:Success**, as shown in the following example:

```
spec:
  kubeletConfig:
    maxPods: 500
```

```

machineConfigPoolSelector:
  matchLabels:
    custom-kubelet: set-max-pods
status:
  conditions:
  - lastTransitionTime: "2021-06-30T17:04:07Z"
    message: Success
    status: "True"
    type: Success

```

2.3. MODIFYING THE NUMBER OF UNAVAILABLE WORKER NODES

By default, only one machine is allowed to be unavailable when applying the kubelet-related configuration to the available worker nodes. For a large cluster, it can take a long time for the configuration change to be reflected. At any time, you can adjust the number of machines that are updating to speed up the process.

Procedure

1. Edit the **worker** machine config pool:

```
$ oc edit machineconfigpool worker
```

2. Add the **maxUnavailable** field and set the value:

```

spec:
  maxUnavailable: <node_count>

```



IMPORTANT

When setting the value, consider the number of worker nodes that can be unavailable without affecting the applications running on the cluster.

2.4. CONTROL PLANE NODE SIZING

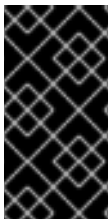
The control plane node resource requirements depend on the number and type of nodes and objects in the cluster. The following control plane node size recommendations are based on the results of a control plane density focused testing, or *Cluster-density*. This test creates the following objects across 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)
27	500	4	16
120	1000	8	32
252	4000	16	64
501	4000	16	96

On a large and dense cluster with three masters or control plane nodes, the CPU and memory usage will spike up when one of the nodes is stopped, rebooted or fails. The failures can be due to unexpected issues with power, network or underlying infrastructure in addition to intentional cases where the cluster is restarted after shutting it down to save costs. The remaining two control plane nodes must handle the load in order to be highly available which leads to increase in the resource usage. This is also expected during upgrades because the masters are cordoned, drained, and rebooted serially to apply the operating system updates, as well as the control plane Operators update. To avoid cascading failures, keep the overall CPU and memory resource usage on the control plane nodes to at most 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 it's memory footprint depends on the number of namespaces and user installed operators that OLM needs to manage on the cluster. Control plane nodes need to be sized accordingly to avoid OOM kills. Following data points are based on the results from cluster maximums testing.

Number of namespaces	OLM memory at idle state (GB)	OLM memory with 5 user operators installed (GB)
500	0.823	1.7
1000	1.2	2.5
1500	1.7	3.2
2000	2	4.4
3000	2.7	5.6

Number of namespaces	OLM memory at idle state (GB)	OLM memory with 5 user operators installed (GB)
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**

If you used an installer-provisioned infrastructure installation method, you cannot modify the control plane node size in a running OpenShift Container Platform 4.11 cluster. Instead, you must estimate your total node count and use the suggested control plane node size during installation.

**IMPORTANT**

The recommendations are based on the data points captured on OpenShift Container Platform clusters with OpenShift SDN as the network plug-in.

**NOTE**

In OpenShift Container Platform 4.11, 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.

2.4.1. Increasing the flavor size of the Amazon Web Services (AWS) master instances

When you have overloaded AWS master nodes in a cluster and the master nodes require more resources, you can increase the flavor size of the master instances.

**NOTE**

It is recommended to backup etcd before increasing the flavor size of the AWS master instances.

Prerequisites

- You have an IPI (installer-provisioned infrastructure) or UPI (user-provisioned infrastructure) cluster on AWS.

Procedure

1. Open the AWS console, fetch the master instances.
2. Stop one master instance.
3. Select the stopped instance, and click **Actions** → **Instance Settings** → **Change instance type**
4. 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**.
5. Backup the instance, and repeat the steps for the next master instance.

Additional resources

- [Backing up etcd](#)

2.5. RECOMMENDED ETCD PRACTICES

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 etcd's consensus protocol 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 a OpenShift API slowness, which affects cluster performance. Because of these reasons, avoid colocating other workloads on the control-plane nodes.

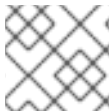
In terms of latency, run etcd on top of a block device that can write at least 50 IOPS of 8000 bytes long sequentially. That is, with a latency of 20ms, keep in mind that uses `fdatasync` to synchronize each write in the WAL. For heavy loaded clusters, sequential 500 IOPS of 8000 bytes (2 ms) are recommended. To measure those numbers, you can use a benchmarking tool, such as `fio`.

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.

The following hard disk features provide optimal etcd performance:

- Low latency to support fast read operation.
- High-bandwidth writes for faster compactions and defragmentation.
- High-bandwidth reads for faster recovery from failures.
- Solid state drives as a minimum selection, however NVMe drives are preferred.
- Server-grade hardware from various manufacturers for increased reliability.
- RAID 0 technology for increased performance.
- Dedicated etcd drives. Do not place log files or other heavy workloads on etcd drives.

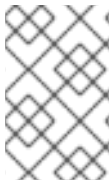
Avoid NAS or SAN setups and spinning drives. Always benchmark by using utilities such as fio. Continuously monitor the cluster performance as it increases.



NOTE

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.

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're 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/openshift-scale/etcd-perf
```

- If you use Docker, run this command:

```
$ sudo docker run --volume /var/lib/etcd:/var/lib/etcd:Z quay.io/openshift-scale/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 20 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](#)

2.6. DEFRAGMENTING ETCD DATA

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

Monitor these key metrics:

- **etcd_server_quota_backend_bytes**, which is the current quota limit
- **etcd_mvcc_db_total_size_in_use_in_bytes**, which indicates the actual database usage after a history compaction
- **etcd_debugging_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.

2.6.1. Automatic defragmentation

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

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

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

**WARNING**

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

Example log output for successful defragmentation

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

Example log output for unsuccessful defragmentation

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

2.6.2. Manual defragmentation

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

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

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

`(etcd_mvcc_db_total_size_in_bytes - etcd_mvcc_db_total_size_in_use_in_bytes)/1024/1024`

**WARNING**

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

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

Prerequisites

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

Procedure

1. Determine which etcd member is the leader, because the leader should be defragmented last.

- a. Get the list of etcd pods:

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

Example output

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

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

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

Example output

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

```
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+
|   ENDPOINT   |   ID   | VERSION | DB SIZE | IS LEADER | IS LEARNER |
| RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+
| https://10.0.191.37:2379 | 251cd44483d811c3 | 3.4.9 | 104 MB | false | false |
7 | 91624 | 91624 | |
| https://10.0.159.225:2379 | 264c7c58ecbdabee | 3.4.9 | 104 MB | false | false |
7 | 91624 | 91624 | |
| https://10.0.199.170:2379 | 9ac311f93915cc79 | 3.4.9 | 104 MB | true | false |
7 | 91624 | 91624 | |
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+
```

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

2. Defragment an etcd member.

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

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

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

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

- c. Defragment the etcd member:

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

Example output

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

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

- d. Verify that the database size was reduced:

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

Example output

```
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+
|   ENDPOINT   |   ID   | VERSION | DB SIZE | IS LEADER | IS LEARNER |
| RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+
| https://10.0.191.37:2379 | 251cd44483d811c3 | 3.4.9 | 104 MB | false | false |
7 | 91624 | 91624 | |
| https://10.0.159.225:2379 | 264c7c58ecbdabee | 3.4.9 | 41 MB | false | false |
7 | 91624 | 91624 | 1
| https://10.0.199.170:2379 | 9ac311f93915cc79 | 3.4.9 | 104 MB | true | false |
7 | 91624 | 91624 | |
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+

```

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

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

- a. Check if there are any **NOSPACE** alarms:

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

Example output

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

- b. Clear the alarms:

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

Next steps

After defragmentation, if etcd still uses more than 50% of its available space, consider increasing the disk quota for etcd.

2.7. OPENSIFT CONTAINER PLATFORM INFRASTRUCTURE COMPONENTS

The following infrastructure workloads do not incur OpenShift Container Platform worker subscriptions:

- Kubernetes and OpenShift Container Platform control plane services that run on masters
- The default router
- The integrated container image registry
- The HAProxy-based Ingress Controller
- The cluster metrics collection, or monitoring service, including components for monitoring user-defined projects
- Cluster aggregated logging
- Service brokers
- Red Hat Quay
- Red Hat OpenShift Data Foundation
- Red Hat Advanced Cluster Manager
- Red Hat Advanced Cluster Security for Kubernetes
- Red Hat OpenShift GitOps
- Red Hat OpenShift Pipelines

Any node that runs any other container, pod, or component is a worker node that your subscription must cover.

For information on infrastructure nodes and which components can run on infrastructure nodes, see the "Red Hat OpenShift control plane and infrastructure nodes" section in the [OpenShift sizing and subscription guide for enterprise Kubernetes](#) document.

2.8. MOVING THE MONITORING SOLUTION

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

Procedure

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

```
apiVersion: v1
kind: ConfigMap
```

```

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

```

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

2. Apply the new config map:

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

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

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

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

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

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

2.9. MOVING THE DEFAULT REGISTRY

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

Prerequisites

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

Procedure

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

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

Example output

```
apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  creationTimestamp: 2019-02-05T13:52:05Z
  finalizers:
  - imageregistry.operator.openshift.io/finalizer
  generation: 1
  name: cluster
  resourceVersion: "56174"
  selfLink: /apis/imageregistry.operator.openshift.io/v1/configs/cluster
  uid: 36fd3724-294d-11e9-a524-12fdee2931b
spec:
  httpSecret: d9a012ccd117b1e6616ceccb2c3bb66a5fed1b5e481623
  logging: 2
  managementState: Managed
  proxy: {}
  replicas: 1
  requests:
    read: {}
    write: {}
  storage:
    s3:
      bucket: image-registry-us-east-1-c92e88cad85b48ec8b312344dff03c82-392c
      region: us-east-1
status:
...
```

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

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

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

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

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

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

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

b. Confirm the node has the label you specified:

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

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

2.10. MOVING THE ROUTER

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

Prerequisites

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

Procedure

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

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

The command output resembles the following text:

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


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

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

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

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

3. Confirm that the router pod is running on the **infra** node.
 - a. View the list of router pods and note the node name of the running pod:

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

Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
NOMINATED NODE READINESS GATES						
router-default-86798b4b5d-bdlvd	1/1	Running	0	28s	10.130.2.4	ip-10-0-217-226.ec2.internal
router-default-955d875f4-255g8	0/1	Terminating	0	19h	10.129.2.4	ip-10-0-148-172.ec2.internal

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

- b. View the node status of the running pod:

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

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

Example output

NAME	STATUS	ROLES	AGE	VERSION
ip-10-0-217-226.ec2.internal	Ready	infra,worker	17h	v1.24.0

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

2.11. 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 of cluster maximums and control plane density focused testing.

Number of worker nodes	CPU cores	Memory (GB)
25	4	16
100	8	32
250	16	128
500	32	128

In general, three infrastructure nodes are recommended per cluster.



IMPORTANT

These sizing recommendations are based on scale tests, which create a large number of objects across the cluster. These tests include reaching some of the cluster maximums. In the case of 250 and 500 node counts on an OpenShift Container Platform 4.11 cluster, these maximums are 10000 namespaces with 61000 pods, 10000 deployments, 181000 secrets, 400 config maps, and so on. 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. The disk size also depends on the retention period. You must take these factors into consideration and size them accordingly.

These sizing recommendations are only applicable for the Prometheus, Router, and Registry infrastructure components, which are installed during cluster installation. Logging is a day-two operation and is not included in these recommendations.



NOTE

In OpenShift Container Platform 4.11, 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.

2.12. ADDITIONAL RESOURCES

- [OpenShift Container Platform cluster maximums](#)
- [Creating infrastructure machine sets](#)

CHAPTER 3. RECOMMENDED HOST PRACTICES FOR IBM Z & LINUXONE ENVIRONMENTS

This topic provides recommended host practices for OpenShift Container Platform on IBM Z and 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 LinuxONE.

3.1. 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](#)

3.2. 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.

3.3. 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.

3.3.1. 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](#)

3.4. 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 SDN 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](#)

3.5. 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.

3.5.1. 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](#)

3.6. 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 LinuxONE environments.

3.6.1. Use multiple queues for your VirtIO network interfaces

With multiple virtual CPUs, you can transfer packages in parallel if you provide multiple queues for incoming and outgoing packets. Use the **queues** attribute of the **driver** element to configure multiple queues. Specify an integer of at least 2 that does not exceed the number of virtual CPUs of the virtual server.

The following example specification configures two input and output queues for a network interface:

```
<interface type="direct">
  <source network="net01"/>
  <model type="virtio"/>
  <driver ... queues="2"/>
</interface>
```

Multiple queues are designed to provide enhanced performance for a network interface, but they also use memory and CPU resources. Start with defining two queues for busy interfaces. Next, try two queues for interfaces with less traffic or more than two queues for busy interfaces.

3.6.2. 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 `<iothreads>3</iothreads>` 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>
  <iothreads>3</iothreads> 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.

3.6.3. 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.

3.6.4. 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>
```

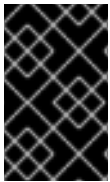
3.6.5. Exclude the memory balloon device

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"/>
```

3.6.6. 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
```

3.6.7. 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 libvirtd
```

- c. Restart the virtual machines.

This setting persists across host reboots.

3.6.8. 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 4. RECOMMENDED CLUSTER SCALING PRACTICES



IMPORTANT

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

These guidelines apply to OpenShift Container Platform with software-defined networking (SDN), not Open Virtual Network (OVN).

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.

4.1. RECOMMENDED PRACTICES FOR SCALING THE CLUSTER

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 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 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 and 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 as 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 QPS and burst rates are currently set to **5** and **10** respectively and they cannot be modified in OpenShift Container Platform.

4.2. MODIFYING A MACHINE SET

To make changes to a machine set, edit the **MachineSet** YAML. Then, remove all machines associated with the machine set by deleting each machine or scaling down the machine set to **0** replicas. Then, scale

the replicas back to the desired number. Changes you make to a machine set do not affect existing machines.

If you need to scale a machine set without making other changes, you do not need to delete the machines.



NOTE

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

Prerequisites

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

Procedure

1. Edit the machine set:

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

2. Scale down the machine set to **0**:

```
$ oc scale --replicas=0 machineset <machineset> -n openshift-machine-api
```

Or:

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

TIP

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

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 0
```

Wait for the machines to be removed.

3. Scale up the machine set as needed:

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

Or:

-

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

TIP

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

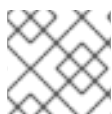
```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 2
```

Wait for the machines to start. The new machines contain changes you made to the machine set.

4.3. ABOUT MACHINE HEALTH CHECKS

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

To monitor machine health, create a resource to define the configuration for a controller. Set a condition to check, such as staying in the **NotReady** status for five minutes or displaying a permanent condition in the node-problem-detector, and a label for the set of machines to monitor.



NOTE

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

The controller that observes a **MachineHealthCheck** resource checks for the defined condition. If a machine fails the health check, the machine is automatically deleted and one is created to take its place. When a machine is deleted, you see a **machine deleted** event.

To limit disruptive impact of the machine deletion, the controller drains and deletes only one node at a time. If there are more unhealthy machines than the **maxUnhealthy** threshold allows for in the targeted pool of machines, remediation stops and therefore enables manual intervention.



NOTE

Consider the timeouts carefully, accounting for workloads and requirements.

- Long timeouts can result in long periods of downtime for the workload on the unhealthy machine.
- Too short timeouts can result in a remediation loop. For example, the timeout for checking the **NotReady** status must be long enough to allow the machine to complete the startup process.

To stop the check, remove the resource.

4.3.1. Limitations when deploying machine health checks

There are limitations to consider before deploying a machine health check:

- Only machines owned by a machine set are remediated by a machine health check.
- Control plane machines are not currently supported and are not remediated if they are unhealthy.
- If the node for a machine is removed from the cluster, a machine health check considers the machine to be unhealthy and remediates it immediately.
- If the corresponding node for a machine does not join the cluster after the **nodeStartupTimeout**, the machine is remediated.
- A machine is remediated immediately if the **Machine** resource phase is **Failed**.

4.4. SAMPLE MACHINEHEALTHCHECK RESOURCE

The **MachineHealthCheck** resource for all cloud-based installation types, and other than bare metal, resembles the following YAML file:

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: example 1
  namespace: openshift-machine-api
spec:
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-machine-role: <role> 2
      machine.openshift.io/cluster-api-machine-type: <role> 3
      machine.openshift.io/cluster-api-machineset: <cluster_name>-<label>-<zone> 4
  unhealthyConditions:
    - type: "Ready"
      timeout: "300s" 5
      status: "False"
    - type: "Ready"
      timeout: "300s" 6
      status: "Unknown"
  maxUnhealthy: "40%" 7
  nodeStartupTimeout: "10m" 8
```

1 Specify the name of the machine health check to deploy.

2 3 Specify a label for the machine pool that you want to check.

4 Specify the machine set to track in **<cluster_name>-<label>-<zone>** format. For example, **prod-node-us-east-1a**.

5 6 Specify the timeout duration for a node condition. If a condition is met for the duration of the timeout, the machine will be remediated. Long timeouts can result in long periods of downtime for a workload on an unhealthy machine.

7 Specify the amount of machines allowed to be concurrently remediated in the targeted pool. This can be set as a percentage or an integer. If the number of unhealthy machines exceeds the limit set

- 8 Specify the timeout duration that a machine health check must wait for a node to join the cluster before a machine is determined to be unhealthy.



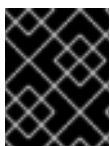
NOTE

The **matchLabels** are examples only; you must map your machine groups based on your specific needs.

4.4.1. Short-circuiting machine health check remediation

Short circuiting ensures that machine health checks remediate machines only when the cluster is healthy. Short-circuiting is configured through the **maxUnhealthy** field in the **MachineHealthCheck** resource.

If the user defines a value for the **maxUnhealthy** field, before remediating any machines, the **MachineHealthCheck** compares the value of **maxUnhealthy** with the number of machines within its target pool that it has determined to be unhealthy. Remediation is not performed if the number of unhealthy machines exceeds the **maxUnhealthy** limit.



IMPORTANT

If **maxUnhealthy** is not set, the value defaults to **100%** and the machines are remediated regardless of the state of the cluster.

The appropriate **maxUnhealthy** value depends on the scale of the cluster you deploy and how many machines the **MachineHealthCheck** covers. For example, you can use the **maxUnhealthy** value to cover multiple machine sets across multiple availability zones so that if you lose an entire zone, your **maxUnhealthy** setting prevents further remediation within the cluster. In global Azure regions that do not have multiple availability zones, you can use availability sets to ensure high availability.

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

4.4.1.1. Setting maxUnhealthy by using an absolute value

If **maxUnhealthy** is set to **2**:

- Remediation will be performed if 2 or fewer nodes are unhealthy
- Remediation will not be performed if 3 or more nodes are unhealthy

These values are independent of how many machines are being checked by the machine health check.

4.4.1.2. Setting maxUnhealthy by using percentages

If **maxUnhealthy** is set to **40%** and there are 25 machines being checked:

- Remediation will be performed if 10 or fewer nodes are unhealthy
- Remediation will not be performed if 11 or more nodes are unhealthy

If **maxUnhealthy** is set to **40%** and there are 6 machines being checked:

- Remediation will be performed if 2 or fewer nodes are unhealthy

- Remediation will not be performed if 3 or more nodes are unhealthy



NOTE

The allowed number of machines is rounded down when the percentage of **maxUnhealthy** machines that are checked is not a whole number.

4.5. CREATING A MACHINEHEALTHCHECK RESOURCE

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

Prerequisites

- Install the **oc** command line interface.

Procedure

1. Create a **healthcheck.yml** file that contains the definition of your machine health check.
2. Apply the **healthcheck.yml** file to your cluster:

```
$ oc apply -f healthcheck.yml
```


CHAPTER 5. 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.

5.1. 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, these functions are part of the Node Tuning Operator.

5.2. 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/default -o yaml -n openshift-cluster-node-tuning-operator
```

The default CR is meant for delivering standard node-level tuning for the OpenShift Container Platform

platform and it can only be modified to set the Operator Management state. Any other custom changes to the default CR will be overwritten by the Operator. For custom tuning, create your own Tuned CRs. Newly created CRs will be combined with the default CR and custom tuning applied to OpenShift Container Platform nodes based on node or pod labels and profile priorities.



WARNING

While in certain situations the support for pod labels can be a convenient way of automatically delivering required tuning, this practice is discouraged and strongly advised against, especially in large-scale clusters. The default Tuned CR ships without pod label matching. If a custom profile is created with pod label matching, then the functionality will be enabled at that time. The pod label functionality will be deprecated in future versions of the Node Tuning Operator.

5.3. DEFAULT PROFILES SET ON A CLUSTER

The following are the default profiles set on a cluster.

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: default
  namespace: openshift-cluster-node-tuning-operator
spec:
  recommend:
  - profile: "openshift-control-plane"
    priority: 30
    match:
    - label: "node-role.kubernetes.io/master"
    - label: "node-role.kubernetes.io/infra"

  - profile: "openshift-node"
    priority: 40
```

Starting with OpenShift Container Platform 4.9, all OpenShift TuneD profiles are shipped with the TuneD package. You can use the **oc exec** command to view the contents of these profiles:

```
$ oc exec $tuned_pod -n openshift-cluster-node-tuning-operator -- find /usr/lib/tuned/openshift{-control-plane,-node} -name tuned.conf -exec grep -H ^ {} \;
```

5.4. VERIFYING THAT THE TUNED PROFILES ARE APPLIED

Verify the TuneD profiles that are applied to your cluster node.

```
$ oc get profile -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.

5.5. 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: ❶
  <mcLabels> ❷
  match: ❸
  <match> ❹
  priority: <priority> ❺
  profile: <tuned_profile_name> ❻
  operand: ❼
  debug: <bool> ❽
  tunedConfig:
    reapply_sysctl: <bool> ❾
```

- ❶ Optional.
- ❷ A dictionary of key/value **MachineConfig** labels. The keys must be unique.
- ❸ If omitted, profile match is assumed unless a profile with a higher priority matches first or **machineConfigLabels** is set.
- ❹ An optional list.
- ❺ Profile ordering priority. Lower numbers mean higher priority (**0** is the highest priority).
- ❻ A Tuned profile to apply on a match. For example **tuned_profile_1**.
- ❼ Optional operand configuration.
- ❽ Turn debugging on or off for the Tuned daemon. Options are **true** for on or **false** for off. The default is **false**.
- ❾ Turn **reapply_sysctl** functionality on or off for the Tuned daemon. Options are **true** for on and **false** for off.

<match> is an optional list recursively defined as follows:

```

- label: <label_name> 1
  value: <label_value> 2
  type: <label_type> 3
  <match> 4

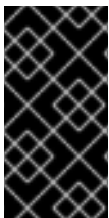
```

- 1 Node or pod label name.
- 2 Optional node or pod label value. If omitted, the presence of **<label_name>** is enough to match.
- 3 Optional object type (**node** or **pod**). If omitted, **node** is assumed.
- 4 An optional **<match>** list.

If **<match>** is not omitted, all nested **<match>** sections must also evaluate to **true**. Otherwise, **false** is assumed and the profile with the respective **<match>** section will not be applied or recommended. Therefore, the nesting (child **<match>** sections) works as logical AND operator. Conversely, if any item of the **<match>** list matches, the entire **<match>** list evaluates to **true**. Therefore, the list acts as logical OR operator.

If **machineConfigLabels** is defined, machine config pool based matching is turned on for the given **recommend:** list item. **<mcLabels>** specifies the labels for a machine config. The machine config is created automatically to apply host settings, such as kernel boot parameters, for the profile **<tuned_profile_name>**. This involves finding all machine config pools with machine config selector matching **<mcLabels>** and setting the profile **<tuned_profile_name>** on all nodes that are assigned the found machine config pools. To target nodes that have both master and worker roles, you must use the master role.

The list items **match** and **machineConfigLabels** are connected by the logical OR operator. The **match** item is evaluated first in a short-circuit manner. Therefore, if it evaluates to **true**, the **machineConfigLabels** item is not considered.



IMPORTANT

When using machine config pool based matching, it is advised to group nodes with the same hardware configuration into the same machine config pool. Not following this practice might result in TuneD operands calculating conflicting kernel parameters for two or more nodes sharing the same machine config pool.

Example: node or pod label based matching

```

- match:
  - label: tuned.openshift.io/elasticsearch
    match:
      - label: node-role.kubernetes.io/master
      - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra
  priority: 20

```

```

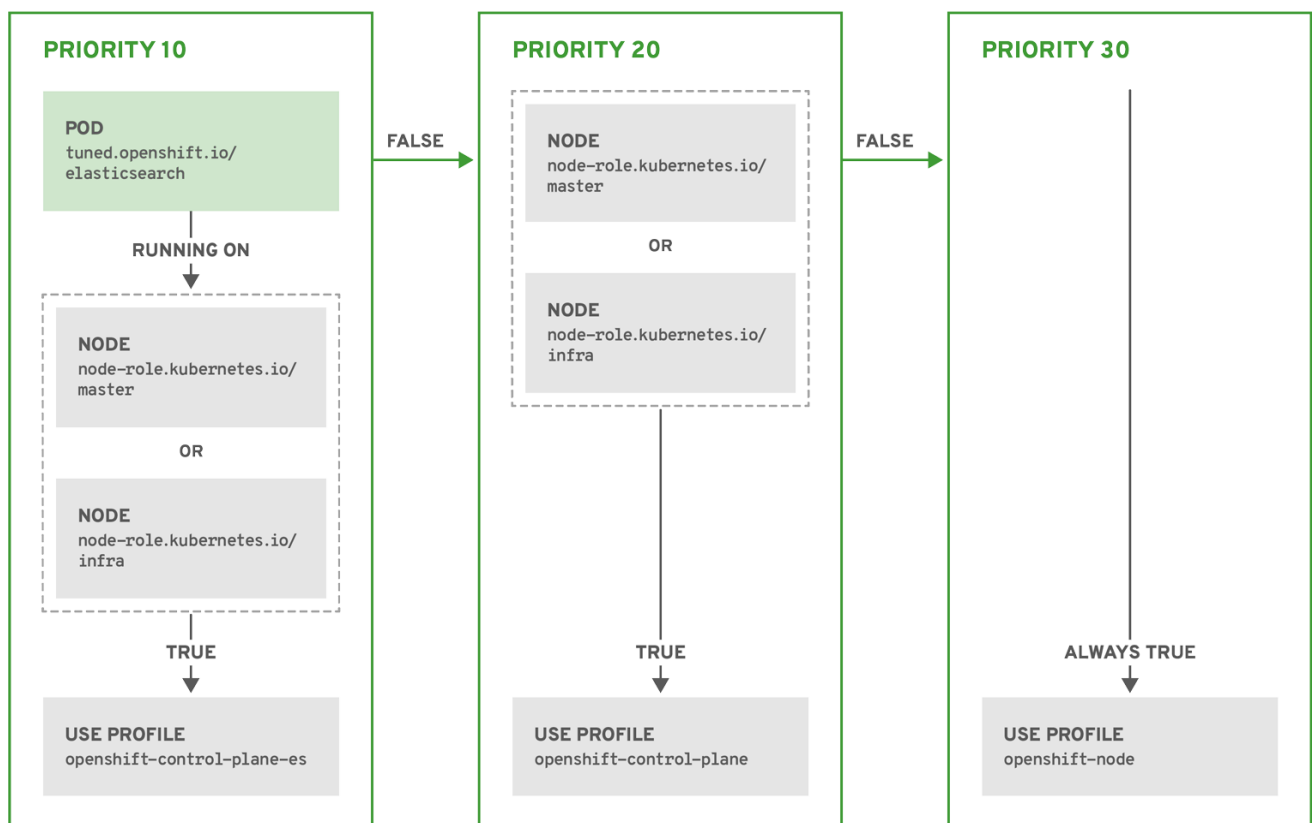
profile: openshift-control-plane
- priority: 30
profile: openshift-node

```

The CR above is translated for the containerized TuneD daemon into its **recommend.conf** file based on the profile priorities. The profile with the highest priority (**10**) is **openshift-control-plane-es** and, therefore, it is considered first. The containerized TuneD daemon running on a given node looks to see if there is a pod running on the same node with the **tuned.openshift.io/elasticsearch** label set. If not, the entire **<match>** section evaluates as **false**. If there is such a pod with the label, in order for the **<match>** section to evaluate to **true**, the node label also needs to be **node-role.kubernetes.io/master** or **node-role.kubernetes.io/infra**.

If the labels for the profile with priority **10** matched, **openshift-control-plane-es** profile is applied and no other profile is considered. If the node/pod label combination did not match, the second highest priority profile (**openshift-control-plane**) is considered. This profile is applied if the containerized TuneD pod runs on a node with labels **node-role.kubernetes.io/master** or **node-role.kubernetes.io/infra**.

Finally, the profile **openshift-node** has the lowest priority of **30**. It lacks the **<match>** section and, therefore, will always match. It acts as a profile catch-all to set **openshift-node** profile, if no other profile with higher priority matches on a given node.



OPENSHIFT_10_0319

Example: machine config pool based matching

```

apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: openshift-node-custom
  namespace: openshift-cluster-node-tuning-operator

```

```
spec:
  profile:
    - data: |
        [main]
        summary=Custom OpenShift node profile with an additional kernel parameter
        include=openshift-node
        [bootloader]
        cmdline_openshift_node_custom=+skew_tick=1
      name: openshift-node-custom

  recommend:
    - machineConfigLabels:
        machineconfiguration.openshift.io/role: "worker-custom"
      priority: 20
      profile: openshift-node-custom
```

To minimize node reboots, label the target nodes with a label the machine config pool's node selector will match, then create the Tuned CR above and finally create the custom machine config pool itself.

Cloud provider-specific Tuned profiles

With this functionality, all Cloud provider-specific nodes can conveniently be assigned a Tuned profile specifically tailored to a given Cloud provider on a OpenShift Container Platform cluster. This can be accomplished without adding additional node labels or grouping nodes into machine config pools.

This functionality takes advantage of **spec.providerID** node object values in the form of **<cloud-provider>://<cloud-provider-specific-id>** and writes the file **/var/lib/tuned/provider** with the value **<cloud-provider>** in NTO operand containers. The content of this file is then used by Tuned to load **provider-<cloud-provider>** profile if such profile exists.

The **openshift** profile that both **openshift-control-plane** and **openshift-node** profiles inherit settings from is now updated to use this functionality through the use of conditional profile loading. Neither NTO nor Tuned currently ship any Cloud provider-specific profiles. However, it is possible to create a custom profile **provider-<cloud-provider>** that will be applied to all Cloud provider-specific cluster nodes.

Example GCE Cloud provider profile

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: provider-gce
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
    - data: |
        [main]
        summary=GCE Cloud provider-specific profile
        # Your tuning for GCE Cloud provider goes here.
      name: provider-gce
```



NOTE

Due to profile inheritance, any setting specified in the **provider-<cloud-provider>** profile will be overwritten by the **openshift** profile and its child profiles.

5.6. 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.

5.7. SUPPORTED TUNED DAEMON PLUG-INS

Excluding the **[main]** section, the following TuneD plug-ins are supported when using custom profiles defined in the **profile:** section of the Tuned CR:

- audio
- cpu
- disk
- eeepc_she
- modules
- mounts
- net
- scheduler
- scsi_host
- selinux
- sysctl
- sysfs
- usb
- video
- vm
- bootloader

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

- `script`
- `systemd`

**WARNING**

The TuneD bootloader plug-in is currently supported on Red Hat Enterprise Linux CoreOS (RHCOS) 8.x worker nodes. For Red Hat Enterprise Linux (RHEL) 7.x worker nodes, the TuneD bootloader plug-in is currently not supported.

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

CHAPTER 6. USING CPU 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.

6.1. SETTING UP CPU MANAGER

Procedure

1. Optional: Label a node:

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

2. Edit the **MachineConfigPool** of the nodes where CPU Manager should be enabled. In this example, all workers have CPU Manager enabled:

```
# oc edit machineconfigpool worker
```

3. Add a label to the worker machine config pool:

```
metadata:
  creationTimestamp: 2020-xx-xxx
  generation: 3
  labels:
    custom-kubelet: cpumanager-enabled
```

4. Create a **KubeletConfig**, **cpumanager-kubeletconfig.yaml**, custom resource (CR). Refer to the label created in the previous step to have the correct nodes updated with the new kubelet config. See the **machineConfigPoolSelector** section:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cpumanager-enabled
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: cpumanager-enabled
  kubeletConfig:
    cpuManagerPolicy: static 1
    cpuManagerReconcilePeriod: 5s 2
```

- 1** Specify a policy:

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

```
# oc create -f cpumanager-kubeletconfig.yaml
```

This adds the CPU Manager feature to the kubelet config and, if needed, the Machine Config Operator (MCO) reboots the node. To enable CPU Manager, a reboot is not needed.

6. Check for the merged kubelet config:

```
# oc get machineconfig 99-worker-XXXXXX-XXXXX-XXXX-XXXXX-kubelet -o json | grep ownerReference -A7
```

Example output

```
"ownerReferences": [
  {
    "apiVersion": "machineconfiguration.openshift.io/v1",
    "kind": "KubeletConfig",
    "name": "cpumanager-enabled",
    "uid": "7ed5616d-6b72-11e9-aae1-021e1ce18878"
  }
]
```

7. Check the worker for the updated **kubelet.conf**:

```
# oc debug node/perf-node.example.com
sh-4.2# cat /host/etc/kubernetes/kubelet.conf | grep cpuManager
```

Example output

```
cpuManagerPolicy: static 1
cpuManagerReconcilePeriod: 5s 2
```

1 2 These settings were defined when you created the **KubeletConfig** CR.

8. Create a pod that requests a core or multiple cores. Both limits and requests must have their CPU value set to a whole integer. That is the number of cores that will be dedicated to this pod:

```
# cat cpumanager-pod.yaml
```

Example output

```

apiVersion: v1
kind: Pod
metadata:
  generateName: cpumanager-
spec:
  containers:
  - name: cpumanager
    image: gcr.io/google_containers/pause-amd64:3.0
    resources:
      requests:
        cpu: 1
        memory: "1G"
      limits:
        cpu: 1
        memory: "1G"
    nodeSelector:
      cpumanager: "true"

```

9. Create the pod:

```
# oc create -f cpumanager-pod.yaml
```

10. Verify that the pod is scheduled to the node that you labeled:

```
# oc describe pod cpumanager
```

Example output

```

Name:          cpumanager-6cqz7
Namespace:     default
Priority:       0
PriorityClassName: <none>
Node: perf-node.example.com/xxx.xx.xx.xxx
...
Limits:
  cpu: 1
  memory: 1G
Requests:
  cpu: 1
  memory: 1G
...
QoS Class:     Guaranteed
Node-Selectors: cpumanager=true

```

11. Verify that the **cgroups** are set up correctly. Get the process ID (PID) of the **pause** process:

```

# └─init.scope
|   └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 17
└─kubepods.slice
    └─kubepods-pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice
        └─crio-b5437308f1a574c542bdf08563b865c0345c8f8c0b0a655612c.scope
            └─32706 /pause

```

Pods of quality of service (QoS) tier **Guaranteed** are placed within the **kubepods.slice**. Pods of other QoS tiers end up in child **cgroups** of **kubepods**:

```
# cd /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-
pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice/crio-
b5437308f1ad1a7db0574c542bdf08563b865c0345c86e9585f8c0b0a655612c.scope
# for i in `ls cpuset.cpus tasks` ; do echo -n "$i "; cat $i ; done
```

Example output

```
cpuset.cpus 1
tasks 32706
```

12. Check the allowed CPU list for the task:

```
# grep ^Cpus_allowed_list /proc/32706/status
```

Example output

```
Cpus_allowed_list: 1
```

13. Verify that another pod (in this case, the pod in the **burstable** QoS tier) on the system cannot run on the core allocated for the **Guaranteed** pod:

```
# cat /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-besteffort.slice/kubepods-besteffort-
podc494a073_6b77_11e9_98c0_06bba5c387ea.slice/crio-
c56982f57b75a2420947f0afc6cafe7534c5734efc34157525fa9abbf99e3849.scope/cpuset.cpus

0
# oc describe node perf-node.example.com
```

Example output

```
...
Capacity:
attachable-volumes-aws-ebs: 39
cpu: 2
ephemeral-storage: 124768236Ki
hugepages-1Gi: 0
hugepages-2Mi: 0
memory: 8162900Ki
pods: 250
Allocatable:
attachable-volumes-aws-ebs: 39
cpu: 1500m
ephemeral-storage: 124768236Ki
hugepages-1Gi: 0
hugepages-2Mi: 0
memory: 7548500Ki
pods: 250
-----
-
default cpumanager-6cqz7 1 (66%) 1 (66%) 1G (12%)
```

1G (12%)	29m
Allocated resources:	
(Total limits may be over 100 percent, i.e., overcommitted.)	
Resource	Requests Limits
-----	-----
cpu	1440m (96%) 1 (66%)

This VM has two CPU cores. The **system-reserved** setting reserves 500 millicores, meaning that half of one core is subtracted from the total capacity of the node to arrive at the **Node Allocatable** amount. You can see that **Allocatable CPU** is 1500 millicores. This means you can run one of the CPU Manager pods since each will take one whole core. A whole core is equivalent to 1000 millicores. If you try to schedule a second pod, the system will accept the pod, but it will never be scheduled:

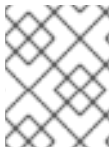
NAME	READY	STATUS	RESTARTS	AGE
cpumanager-6cqz7	1/1	Running	0	33m
cpumanager-7qc2t	0/1	Pending	0	11s

CHAPTER 7. USING TOPOLOGY MANAGER

Topology Manager collects hints from the CPU Manager, Device Manager, and other Hint Providers to align pod resources, such as CPU, SR-IOV VFs, 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 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.



NOTE

To use Topology Manager you must use the CPU Manager with the **static** policy. For more information on CPU Manager, see [Using CPU Manager](#).

7.1. TOPOLOGY MANAGER POLICIES

Topology Manager aligns **Pod** resources of all Quality of Service (QoS) classes by collecting topology hints from Hint Providers, such as CPU Manager and Device Manager, and using the collected hints to align the **Pod** resources.



NOTE

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

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

none policy

This is the default policy and does not perform any topology alignment.

best-effort policy

For each container in a pod with the **best-effort** topology management policy, kubelet calls each Hint Provider to discover their resource availability. Using this information, the Topology Manager stores the preferred NUMA Node affinity for that container. If the affinity is not preferred, Topology Manager stores this and admits the pod to the node.

restricted policy

For each container in a pod with the **restricted** topology management policy, kubelet calls each Hint Provider to discover their resource availability. Using this information, the Topology Manager stores the preferred NUMA Node affinity for that container. If the affinity is not preferred, Topology Manager rejects this pod from the node, resulting in a pod in a **Terminated** state with a pod admission failure.

single-numa-node policy

For each container in a pod with the **single-numa-node** topology management policy, kubelet calls each Hint Provider to discover their resource availability. Using this information, the Topology Manager determines if a single NUMA Node affinity is possible. If it is, the pod is admitted to the node. If a single NUMA Node affinity is not possible, the Topology Manager rejects the pod from the node. This results in a pod in a Terminated state with a pod admission failure.

7.2. SETTING UP TOPOLOGY MANAGER

To use Topology Manager, you must configure an allocation policy in the **cpumanager-enabled** custom resource (CR). This file might exist if you have set up CPU Manager. If the file does not exist, you can create the file.

Prerequisites

- Configure the CPU Manager policy to be **static**. See the Using CPU Manager in the Scalability and Performance section.

Procedure

To activate Topology Manager:

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

```
$ oc edit KubeletConfig cpumanager-enabled
```

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cpumanager-enabled
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: cpumanager-enabled
  kubeletConfig:
    cpuManagerPolicy: static 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**.

Additional resources

- For more information on CPU Manager, see [Using CPU Manager](#).

7.3. POD INTERACTIONS WITH TOPOLOGY MANAGER POLICIES

The example **Pod** specs below help illustrate pod interactions with Topology Manager.

The following pod runs in the **BestEffort** QoS class because no resource requests or limits are specified.

```
spec:
  containers:
  - name: nginx
    image: nginx
```

The next pod runs in the **Burstable** QoS class because requests are less than limits.

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
      requests:
        memory: "100Mi"
```

If the selected policy is anything other than **none**, Topology Manager would not consider either of these **Pod** specifications.

The last example pod below runs in the Guaranteed QoS class because requests are equal to limits.

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
      requests:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
```

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

Topology Manager will use this information to store the best Topology for this container. In the case of this pod, CPU Manager and Device Manager will use this stored information at the resource allocation stage.

CHAPTER 8. 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.



IMPORTANT

NUMA-aware scheduling is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see <https://access.redhat.com/support/offerings/techpreview/>.

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.

8.1. ABOUT NUMA-AWARE SCHEDULING

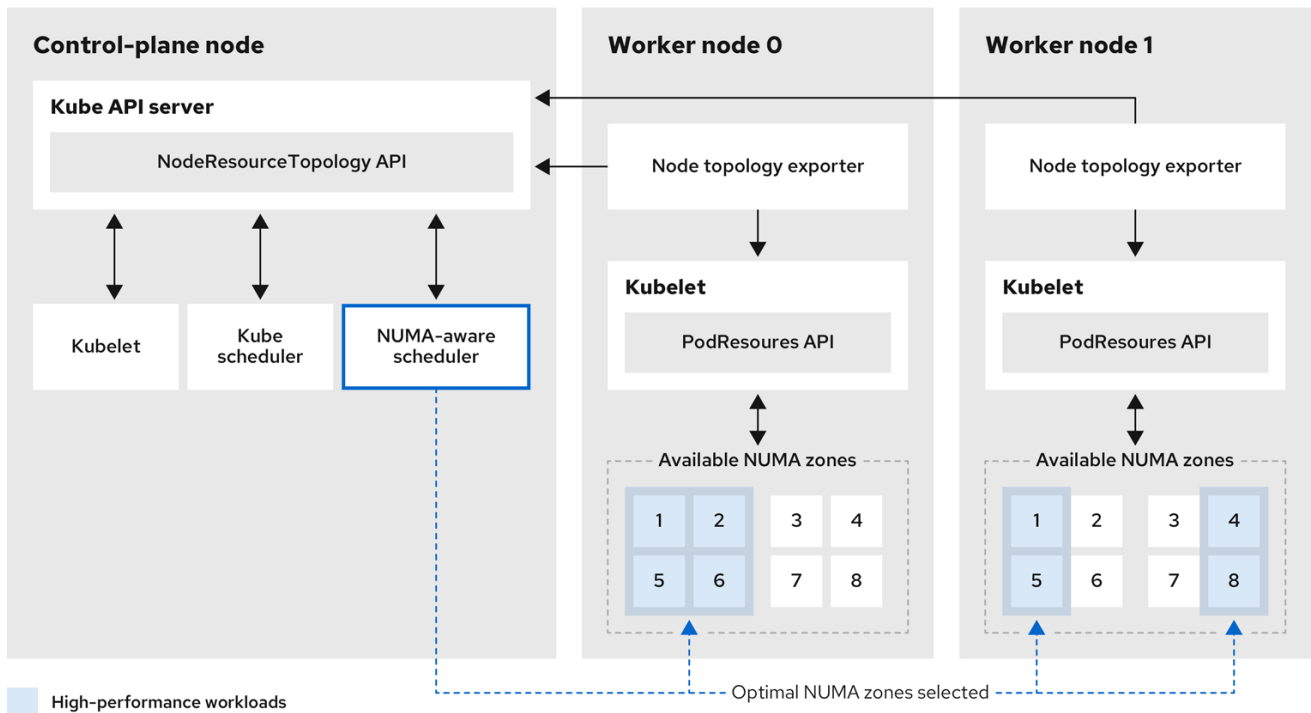
Non-Uniform Memory Access (NUMA) is a compute platform architecture that allows different CPUs to access different regions of memory at different speeds. NUMA resource topology refers to the locations of CPUs, memory, and PCI devices relative to each other in the compute node. Co-located resources are said to be in the same *NUMA zone*. For high-performance applications, the cluster needs to process pod workloads in a single NUMA zone.

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 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.

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 by not 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.

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 8.1. 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.

Additional resources

- For more information about running secondary pod schedulers in your cluster and how to deploy pods with a secondary pod scheduler, see [Scheduling pods using a secondary scheduler](#).

8.2. 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.

8.2.1. 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: "4.11"
  name: numaresources-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

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

```
$ oc create -f nro-sub.yaml
```

Verification

1. 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.v4.11.2	numaresources-operator	4.11.2		Succeeded

8.2.2. 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

1. Install the NUMA Resources Operator using the OpenShift Container Platform web console:
 - a. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
 - b. Choose **NUMA Resources Operator** from the list of available Operators, and then click **Install**.
2. Optional: Verify that the NUMA Resources Operator installed successfully:
 - a. Switch to the **Operators → Installed Operators** page.
 - b. Ensure that **NUMA Resources Operator** is listed in the **default** project 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.

8.3. 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 **MachineConfigPool** custom resource that enables custom kubelet configurations for worker nodes:

- a. Save the following YAML in the **nro-machineconfig.yaml** file:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  labels:
    cnf-worker-tuning: enabled
    machineconfiguration.openshift.io/mco-built-in: ""
    pools.operator.machineconfiguration.openshift.io/worker: ""
  name: worker
spec:
  machineConfigSelector:
    matchLabels:
      machineconfiguration.openshift.io/role: worker
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker: ""
```

- b. Create the **MachineConfigPool** CR by running the following command:

```
$ oc create -f nro-machineconfig.yaml
```

2. Create the **NUMAResourcesOperator** custom resource:

- a. Save the following YAML in the **nrop.yaml** file:

```
apiVersion: nodetopology.openshift.io/v1alpha1
kind: NUMAResourcesOperator
metadata:
  name: numaresourcesoperator
spec:
  nodeGroups:
    - machineConfigPoolSelector:
        matchLabels:
          pools.operator.machineconfiguration.openshift.io/worker: "" 1
```

- 1** Should match the label applied to worker nodes in the related **MachineConfigPool** CR.

- b. Create the **NUMAResourcesOperator** CR by running the following command:

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

Verification

Verify that the NUMA Resources Operator deployed successfully by running the following command:

```
$ oc get numaresourcesoperators.nodetopology.openshift.io
```

Example output

```
NAME                AGE
numaresourcesoperator 10m
```

8.4. DEPLOYING THE NUMA-AWARE SECONDARY POD SCHEDULER

After you install the NUMA Resources Operator, do the following to deploy the NUMA-aware secondary pod scheduler:

- Configure the pod admittance policy for the required machine profile
- Create the required machine config pool
- Deploy the NUMA-aware secondary scheduler

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the NUMA Resources Operator.

Procedure

1. Create the **KubeletConfig** custom resource 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: cnf-worker-tuning
spec:
  machineConfigPoolSelector:
    matchLabels:
      cnf-worker-tuning: enabled
  kubeletConfig:
    cpuManagerPolicy: "static" 1
    cpuManagerReconcilePeriod: "5s"
    reservedSystemCPUs: "0,1"
    memoryManagerPolicy: "Static" 2
```



```

evictionHard:
  memory.available: "100Mi"
kubeReserved:
  memory: "512Mi"
reservedMemory:
  - numaNode: 0
  limits:
    memory: "1124Mi"
systemReserved:
  memory: "512Mi"
topologyManagerPolicy: "single-numa-node" 3
topologyManagerScope: "pod"

```

- 1 For **cpuManagerPolicy**, **static** must use a lowercase **s**.
- 2 For **memoryManagerPolicy**, **Static** must use an uppercase **S**.
- 3 **topologyManagerPolicy** must be set to **single-numa-node**.

b. Create the **KubeletConfig** custom resource (CR) by running the following command:

```
$ oc create -f nro-kubeletconfig.yaml
```

2. Create the **NUMAResourcesScheduler** custom resource that deploys the NUMA-aware custom pod scheduler:

a. Save the following YAML in the **nro-scheduler.yaml** file:

```

apiVersion: nodetopology.openshift.io/v1alpha1
kind: NUMAResourcesScheduler
metadata:
  name: numaresourcesscheduler
spec:
  imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-container-
rhel8:v4.11"

```

b. Create the **NUMAResourcesScheduler** CR by running the following command:

```
$ oc create -f nro-scheduler.yaml
```

Verification

Verify that the required resources deployed successfully by running the following command:

```
$ oc get all -n openshift-numaresources
```

Example output

NAME	READY	STATUS	RESTARTS	AGE
pod/numaresources-controller-manager-7575848485-bns4s	1/1	Running	0	13m
pod/numaresourcesoperator-worker-dvj4n	2/2	Running	0	16m
pod/numaresourcesoperator-worker-lcg4t	2/2	Running	0	16m
pod/secondary-scheduler-56994cf6cf-7qf4q	1/1	Running	0	16m

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE
------	---------	---------	-------	------------	-----------

```

NODE SELECTOR          AGE
daemonset.apps/numaresourcesoperator-worker 2      2      2      2      2      node-
role.kubernetes.io/worker= 16m
NAME                  READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/numaresources-controller-manager 1/1    1          1          13m
deployment.apps/secondary-scheduler             1/1    1          1          16m
NAME                  DESIRED  CURRENT  READY  AGE
replicaset.apps/numaresources-controller-manager-7575848485 1      1      1      13m
replicaset.apps/secondary-scheduler-56994cf6cf              1      1      1      16m

```

8.5. SCHEDULING WORKLOADS WITH THE NUMA-AWARE SCHEDULER

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.
- Install the NUMA Resources Operator and deploy the NUMA-aware secondary scheduler.

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: ctrn
    image: quay.io/openshifttest/hello-openshift:openshift
    imagePullPolicy: IfNotPresent
    resources:
      limits:
        memory: "100Mi"
        cpu: "10"
      requests:
        memory: "100Mi"
        cpu: "10"
  - name: ctrn2
    image: gcr.io/google_containers/pause-amd64:3.0
    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

1. Verify that the deployment was successful:

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

Example output

NAME	READY	STATUS	RESTARTS	AGE
numa-deployment-1-56954b7b46-pfgw8	2/2	Running	0	129m
numaresources-controller-manager-7575848485-bns4s	1/1	Running	0	15h
numaresourcesoperator-worker-dvj4n	2/2	Running	0	18h
numaresourcesoperator-worker-lcg4t	2/2	Running	0	16h
secondary-scheduler-56994cf6cf-7qf4q	1/1	Running	0	18h

2. Verify that the **topo-aware-scheduler** is scheduling the deployed pod by running the following command:

```
$ oc describe pod numa-deployment-1-56954b7b46-pfgw8 -n openshift-numaresources
```

Example output

```
Events:
  Type    Reason      Age    From          Message
  ----    -
  Normal  Scheduled   130m   topo-aware-scheduler  Successfully assigned openshift-numaresources/numa-deployment-1-56954b7b46-pfgw8 to compute-0.example.com
```



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. Run the following command:

```
$ oc describe noderesourcetopologies.topology.node.k8s.io
```

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**.

4. 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** by running the following command:

```
$ oc get pod <pod_name> -n <pod_namespace> -o jsonpath="{ .status.qosClass }"
```

Example output

```
Guaranteed
```

8.6. 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 scheduable 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.

4. Verify the NUMA zone granularity for all scheduable 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/v1alpha1
  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/v1alpha1
  kind: NodeResourceTopology
  metadata:
    annotations:
      k8stopoaware SchedWg/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: ❶
- costs:
  - name: node-0
    value: 10
  - name: node-1
    value: 21
  name: node-0
  resources: ❷
  - 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.

8.6.1. 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/v1alpha1
kind: NUMAResourcesScheduler
metadata:
  name: numaresourcesscheduler
spec:
  imageSpec: "registry.redhat.io/openshift4/noderesourcetopology-scheduler-container-rhel8:v4.11"
  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

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 shows 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"
```

8.6.2. 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
```

- 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
```

8.6.3. 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
```

CHAPTER 9. SCALING THE CLUSTER MONITORING OPERATOR

OpenShift Container Platform exposes metrics that the Cluster Monitoring Operator 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**.

9.1. PROMETHEUS DATABASE STORAGE REQUIREMENTS

Red Hat performed various tests for different scale sizes.



NOTE

The Prometheus storage requirements below are not prescriptive. Higher resource consumption might be observed in your cluster depending on workload activity and resource use.

Table 9.1. Prometheus Database storage requirements based on number of nodes/pods in the cluster

Number of Nodes	Number of pods	Prometheus storage growth per day	Prometheus storage growth per 15 days	RAM Space (per scale size)	Network (per tsdb chunk)
50	1800	6.3 GB	94 GB	6 GB	16 MB
100	3600	13 GB	195 GB	10 GB	26 MB
150	5400	19 GB	283 GB	12 GB	36 MB
200	7200	25 GB	375 GB	14 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 three infrastructure (infra) nodes.
- Use at least three **openshift-container-storage** nodes with non-volatile memory express (NVMe) drives.

9.2. 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 A typical value 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 as 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 as 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
```

CHAPTER 10. 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.



IMPORTANT

These guidelines apply to OpenShift Container Platform with software-defined networking (SDN), not Open Virtual Network (OVN).

In most cases, exceeding these numbers results in lower overall performance. It does not necessarily mean that the cluster will fail.

10.1. OPENSIFT CONTAINER PLATFORM TESTED CLUSTER MAXIMUMS FOR MAJOR RELEASES

Tested Cloud Platforms for OpenShift Container Platform 3.x: Red Hat OpenStack Platform (RHOSP), Amazon Web Services and Microsoft Azure. Tested Cloud Platforms for OpenShift Container Platform 4.x: Amazon Web Services, Microsoft Azure and Google Cloud Platform.

Maximum type	3.x tested maximum	4.x tested maximum
Number of nodes	2,000	2,000 ^[1]
Number of pods ^[2]	150,000	150,000
Number of pods per node	250	500 ^[3]
Number of pods per core	There is no default value.	There is no default value.
Number of namespaces ^[4]	10,000	10,000
Number of builds	10,000 (Default pod RAM 512 Mi) - Pipeline Strategy	10,000 (Default pod RAM 512 Mi) - Source-to-Image (S2I) build strategy
Number of pods per namespace ^[5]	25,000	25,000
Number of routes and back ends per Ingress Controller	2,000 per router	2,000 per router
Number of secrets	80,000	80,000

Maximum type	3.x tested maximum	4.x tested maximum
Number of config maps	90,000	90,000
Number of services ^[6]	10,000	10,000
Number of services per namespace	5,000	5,000
Number of back-ends per service	5,000	5,000
Number of deployments per namespace ^[5]	2,000	2,000
Number of build configs	12,000	12,000
Number of custom resource definitions (CRD)	There is no default value.	512 ^[7]

1. Pause pods were deployed to stress the control plane components of OpenShift Container Platform at 2000 node scale.
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 100 worker nodes with 500 pods per worker node. The default **maxPods** is still 250. To get to 500 **maxPods**, the cluster must be created with a **maxPods** set to **500** using a custom kubelet config. If you need 500 user pods, you need a **hostPrefix** of **22** because there are 10-15 system pods already running on the node. The maximum number of pods with attached persistent volume claims (PVC) depends on storage backend from where PVC are allocated. In our tests, only OpenShift Data Foundation v4 (OCS v4) was able to satisfy the number of pods per node discussed in this document.
4. When there are a large number of active projects, etcd might suffer from poor performance if the key space grows excessively large and exceeds the space quota. Periodic maintenance of etcd, including defragmentation, is highly recommended to free etcd storage.
5. There are a number of 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. OpenShift Container Platform has a limit of 512 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 512 CRDs created, then there is a possibility that **oc** commands requests may be throttled.

**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.

10.2. OPENSIFT CONTAINER PLATFORM ENVIRONMENT AND CONFIGURATION ON WHICH THE CLUSTER MAXIMUMS ARE TESTED

10.2.1. AWS cloud platform

Node	Flavor	vCPU	RAM(GiB)	Disk type	Disk size(GiB) /IOS	Count	Region
Control plane/etcd ^[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	3/25/250 /500 ^[5]	us-west-2

1. gp3 disks with a baseline performance of 3000 IOPS and 125 MiB per second are used for control plane/etcd 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.

10.2.2. IBM Power platform

Node	vCPU	RAM(GiB)	Disk type	Disk size(GiB)/IOS	Count
Control plane/etcd ^[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/etcd 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.

10.2.3. IBM Z platform

Node	vCPU [4]	RAM(GiB)[5]	Disk type	Disk size(GiB)/IOS	Count
Control plane/etcd ^[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/etcd nodes as etcd 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 instantiated. 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.

10.3. 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:

$$\text{required pods per cluster} / \text{pods per node} = \text{total number of nodes needed}$$

The current 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:

$$\text{required pods per cluster} / \text{total number of nodes} = \text{expected pods per node}$$

10.4. 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
    portName: ""
    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 KiB** 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 11. 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.

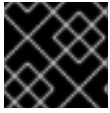
11.1. AVAILABLE PERSISTENT STORAGE OPTIONS

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

Table 11.1. Available storage options

Storage type	Description	Examples
Block	<ul style="list-style-type: none"> Presented to the operating system (OS) as a block device Suitable for applications that need full control of storage and operate at a low level on files bypassing the file system Also referred to as a Storage Area Network (SAN) Non-shareable, which means that only one client at a time can mount an endpoint of this type 	AWS EBS and VMware vSphere support dynamic persistent volume (PV) provisioning natively in OpenShift Container Platform.
File	<ul style="list-style-type: none"> Presented to the OS as a file system export to be mounted Also referred to as Network Attached Storage (NAS) Concurrency, latency, file locking mechanisms, and other capabilities vary widely between protocols, implementations, vendors, and scales. 	RHEL NFS, NetApp NFS ^[1] , and Vendor NFS
Object	<ul style="list-style-type: none"> Accessible through a REST API endpoint Configurable for use in the OpenShift Container Platform Registry Applications must build their drivers into the application and/or container. 	AWS S3

1. NetApp NFS supports dynamic PV provisioning when using the Trident plug-in.

**IMPORTANT**

Currently, CNS is not supported in OpenShift Container Platform 4.11.

11.2. RECOMMENDED CONFIGURABLE STORAGE TECHNOLOGY

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

Table 11.2. Recommended and configurable storage technology

Storage type	ROX ¹	RWX ²	Registry	Scaled registry	Metrics ³	Logging	Apps
Block	Yes ⁴	No	Configurable	Not configurable	Recommended	Recommended	Recommended
File	Yes ⁴	Yes	Configurable	Configurable	Configurable ⁵	Configurable ⁶	Recommended
Object	Yes	Yes	Recommended	Recommended	Not configurable	Not configurable	Not configurable ⁷

¹ **ReadOnlyMany**

² **ReadWriteMany**

³ Prometheus is the underlying technology used for metrics.

⁴ This does not apply to physical disk, VM physical disk, VMDK, loopback over NFS, AWS EBS, and Azure Disk.

⁵ For metrics, using file storage with the **ReadWriteMany** (RWX) access mode is unreliable. If you use file storage, do not configure the RWX access mode on any persistent volume claims (PVCs) that are configured for use with metrics.

⁶ For logging, using any shared storage would be an anti-pattern. One volume per elasticsearch is required.

⁷ Object storage is not consumed through OpenShift Container Platform's PVs or PVCs. Apps must integrate with the object storage REST API.

**NOTE**

A scaled registry is an OpenShift Container Platform registry where two or more pod replicas are running.

11.2.1. Specific application storage recommendations

**IMPORTANT**

Testing shows issues with using the NFS server on Red Hat Enterprise Linux (RHEL) as storage backend for core services. This includes the OpenShift Container Registry and Quay, Prometheus for monitoring storage, and Elasticsearch for logging storage. Therefore, using RHEL NFS to back PVs used by core services is not recommended.

Other NFS implementations on the marketplace might not have these issues. Contact the individual NFS implementation vendor for more information on any testing that was possibly completed against these OpenShift Container Platform core components.

11.2.1.1. Registry

In a non-scaled/high-availability (HA) OpenShift Container Platform registry cluster deployment:

- The storage technology does not have to support RWX access mode.
- The storage technology must ensure read-after-write consistency.
- The preferred storage technology is object storage followed by block storage.
- File storage is not recommended for OpenShift Container Platform registry cluster deployment with production workloads.

11.2.1.2. Scaled registry

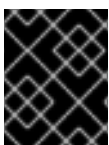
In a scaled/HA OpenShift Container Platform registry cluster deployment:

- The storage technology must support RWX access mode.
- The storage technology must ensure read-after-write consistency.
- The preferred storage technology is object storage.
- Amazon Simple Storage Service (Amazon S3), Google Cloud Storage (GCS), Microsoft Azure Blob Storage, and OpenStack Swift are supported.
- Object storage should be S3 or Swift compliant.
- For non-cloud platforms, such as vSphere and bare metal installations, the only configurable technology is file storage.
- Block storage is not configurable.

11.2.1.3. 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.

11.2.1.4. Logging

In an OpenShift Container Platform hosted logging cluster deployment:

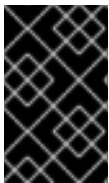
- The preferred storage technology is block storage.
- Object storage is not configurable.

11.2.1.5. 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.

11.2.2. 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*.

11.3. DATA STORAGE MANAGEMENT

The following table summarizes the main directories that OpenShift Container Platform components write data to.

Table 11.3. Main directories for storing OpenShift Container Platform data

Directory	Notes	Sizing	Expected growth
<code>/var/log</code>	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
<i>/var/lib/etcd</i>	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.
<i>/var/lib/containers</i>	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.
<i>/var/lib/kubelet</i>	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.
<i>/var/log</i>	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.

CHAPTER 12. OPTIMIZING ROUTING

The OpenShift Container Platform HAProxy router scales to optimize performance.

12.1. BASELINE INGRESS CONTROLLER (ROUTER) PERFORMANCE

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

When evaluating a single HAProxy router performance in terms of HTTP requests handled per second, the performance varies depending on many factors. In particular:

- HTTP keep-alive/close mode
- Route type
- TLS session resumption client support
- Number of concurrent connections per target route
- Number of target routes
- Back end server page size
- Underlying infrastructure (network/SDN solution, CPU, and so on)

While performance in your specific environment will vary, Red Hat lab tests on a public cloud instance of size 4 vCPU/16GB RAM. A single HAProxy router handling 100 routes terminated by backends serving 1kB static pages is able to handle the following number of transactions per second.

In HTTP keep-alive mode scenarios:

Encryption	LoadBalancerService	HostNetwork
none	21515	29622
edge	16743	22913
passthrough	36786	53295
re-encrypt	21583	25198

In HTTP close (no keep-alive) scenarios:

Encryption	LoadBalancerService	HostNetwork
none	5719	8273
edge	2729	4069
passthrough	4121	5344

Encryption	LoadBalancerService	HostNetwork
re-encrypt	2320	2941

Default Ingress Controller configuration with **ROUTER_THREADS=4** was used and two different endpoint publishing strategies (LoadBalancerService/HostNetwork) were tested. TLS session resumption was used for encrypted routes. With HTTP keep-alive, a single HAProxy router is capable of saturating 1 Gbit NIC at page sizes as small as 8 kB.

When running on bare metal with modern processors, you can expect roughly twice the performance of the public cloud instance above. This overhead is introduced by the virtualization layer in place on public clouds and holds mostly true for private cloud-based virtualization as well. The following table is a guide to how many applications to use behind the router:

Number of applications	Application type
5-10	static file/web server or caching proxy
100-1000	applications generating dynamic content

In general, HAProxy can support routes for 5 to 1000 applications, depending on the technology in use. Ingress Controller performance might be limited by the capabilities and performance of the applications behind it, such as language or static versus dynamic content.

Ingress, or router, sharding should be used to serve more routes towards applications and help horizontally scale the routing tier.

For more information on Ingress sharding, see [Configuring Ingress Controller sharding by using route labels](#) and [Configuring Ingress Controller sharding by using namespace labels](#).

12.2. INGRESS CONTROLLER (ROUTER) PERFORMANCE OPTIMIZATIONS

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

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

12.2.1. 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 short for the kubelet's probes to succeed in some scenarios. Probe timeouts can cause unwanted router restarts that interrupt application connections. The ability to set larger timeout values can reduce the risk of unnecessary and unwanted restarts.

You can update the **timeoutSeconds** value on the **livenessProbe**, **readinessProbe**, and **startupProbe** parameters of the router container.

Parameter	Description
livenessProbe	The livenessProbe reports to the kubelet whether a pod is dead and needs to be restarted.
readinessProbe	The readinessProbe reports whether a pod is healthy or unhealthy. When the readiness probe reports an unhealthy pod, then the kubelet marks the pod as not ready to accept traffic. Subsequently, the endpoints for that pod are marked as not ready, and this status propagates to the kube-proxy. On cloud platforms with a configured load balancer, the kube-proxy communicates to the cloud load-balancer not to send traffic to the node with that pod.
startupProbe	The startupProbe gives the router pod up to 2 minutes to initialize before the kubelet begins sending the router liveness and readiness probes. This initialization time can prevent routers with many routes or endpoints from prematurely restarting.



IMPORTANT

The timeout configuration option is an advanced tuning technique that can be used to work around issues. However, these issues should eventually be diagnosed and possibly a support case or [Jira issue](#) opened for any issues that causes probes to time out.

The following example demonstrates how you can directly patch the default router deployment to set a 5-second timeout for the liveness and readiness probes:

```
$ oc -n openshift-ingress patch deploy/router-default --type=strategic --patch='{"spec":{"template":
{"spec":{"containers":[{"name":"router","livenessProbe":{"timeoutSeconds":5},"readinessProbe":
{"timeoutSeconds":5}]}}}}'
```

Verification

```
$ oc -n openshift-ingress describe deploy/router-default | grep -e Liveness: -e Readiness:
Liveness: http-get http://:1936/healthz delay=0s timeout=5s period=10s #success=1 #failure=3
Readiness: http-get http://:1936/healthz/ready delay=0s timeout=5s period=10s #success=1
#failure=3
```

CHAPTER 13. OPTIMIZING NETWORKING

The [OpenShift SDN](#) uses OpenvSwitch, virtual extensible LAN (VXLAN) tunnels, OpenFlow rules, and iptables. This network can be tuned by using jumbo frames, network interface controllers (NIC) offloads, multi-queue, and ethtool settings.

[OVN-Kubernetes](#) uses Geneve (Generic Network Virtualization Encapsulation) instead of VXLAN as the tunnel protocol.

VXLAN 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.

VXLAN encapsulates all tunneled traffic in user datagram protocol (UDP) packets. However, this leads to increased CPU utilization. Both these outer- and inner-packets are subject to normal checksumming rules to guarantee data is not corrupted during transit. Depending on CPU performance, this additional processing overhead can cause a reduction in throughput and increased latency when compared to traditional, non-overlay networks.

Cloud, VM, and bare metal CPU performance can be capable of handling much more than one Gbps network throughput. When using higher bandwidth links such as 10 or 40 Gbps, reduced performance can occur. This is a known issue in VXLAN-based environments and is not specific to containers or OpenShift Container Platform. Any network that relies on VXLAN tunnels will perform similarly because of the VXLAN implementation.

If you are looking to push beyond one Gbps, you can:

- Evaluate network plug-ins that implement different routing techniques, such as border gateway protocol (BGP).
- Use VXLAN-offload capable network adapters. VXLAN-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 utilize the full bandwidth of their network infrastructure.

VXLAN-offload does not reduce latency. However, CPU utilization is reduced even in latency tests.

13.1. 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 only configured at the time of OpenShift Container Platform installation. The MTU must be less than or equal to the maximum supported value of the NIC of your network. If you are optimizing for throughput, choose the largest possible value. If you are optimizing for lowest latency, choose a lower value.

The SDN overlay's MTU must be less than the NIC MTU by 50 bytes at a minimum. This accounts for the SDN overlay header. So, on a normal ethernet network, set this to **1450**. On a jumbo frame ethernet network, set this to **8950**.

For OVN and Geneve, the MTU must be less than the NIC MTU by 100 bytes at a minimum.

**NOTE**

This 50 byte overlay header is relevant to the OpenShift SDN. Other SDN solutions might require the value to be more or less.

13.2. 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:

```
networking:
  clusterNetwork:
    - cidr: 10.128.0.0/14
      hostPrefix: 23
  machineNetwork:
    - cidr: 10.0.0.0/16
  networkType: OpenShiftSDN
  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. It 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. 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 will lead to decreased throughput and increased CPU usage.

13.4. ADDITIONAL RESOURCES

- [Modifying advanced network configuration parameters](#)
- [Configuration parameters for the OVN-Kubernetes default CNI network provider](#)
- [Configuration parameters for the OpenShift SDN default CNI network provider](#)
- [Improving cluster stability in high latency environments using worker latency profiles](#)

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 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 machine sets describe infrastructure components specific to your configuration. You apply specific Kubernetes labels to these 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 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. Optional: Enable power management for the host. This allows OpenShift Container Platform to control the power state of the host.
7. Enter the user credentials for the host's baseboard management controller (BMC).
8. Select to power on the host after creation, and select **Create**.
9. 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 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>
    disableCertificateVerification: True
  bootMACAddress: <host_boot_mac_address>
  hardwareProfile: unknown
```

- 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](#).

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 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 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 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.

Additional resources

- [Expanding the cluster](#)
- [MachineHealthChecks on bare metal](#)

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 ❶
memory: "1Gi"
cpu: "1"
volumes:
- name: hugepage
  emptyDir:
    medium: HugePages

```

- ❶ 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 plug-in, 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 <.>
        valueFrom:
          resourceFieldRef:
            containerName: example
            resource: requests.hugepages-1Gi
      volumes:
      - name: hugepage
        emptyDir:
          medium: HugePages
      - name: podinfo
        downwardAPI:
          items:
          - path: "hugepages_1G_request" <.>
            resourceFieldRef:
              containerName: example
              resource: requests.hugepages-1Gi
            divisor: 1Gi

```

<.> Specifies to read the resource use from **requests.hugepages-1Gi** and expose the value as the **REQUESTS_HUGEPAGES_1Gi** environment variable. <.> 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_HUGE_PAGES_1Gi
```

Example output

```
REQUESTS_HUGE_PAGES_1Gi=2147483648
```

2. 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

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.

15.4.1. At boot time

Procedure

To minimize node reboots, the order of the steps below needs to be followed:

1. Label all nodes that need the same huge pages setting by a label.

```
$ oc label node <node_using_hugepages> node-role.kubernetes.io/worker-hp=
```

2. Create a file with the following content and name it **hugepages-tuned-boottime.yaml**:

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: hugepages 1
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile: 2
  - data: |
    [main]
    summary=Boot time configuration for hugepages
    include=openshift-node
```

```
[bootloader]
cmdline_openshift_node_hugepages=hugepagesz=2M hugepages=50 3
name: openshift-node-hugepages

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

- 1 Set the **name** of the Tuned resource to **hugepages**.
- 2 Set the **profile** section to allocate huge pages.
- 3 Note the order of parameters is important as some platforms support huge pages of various sizes.
- 4 Enable machine config pool based matching.

3. Create the Tuned **hugepages** object

```
$ oc create -f hugepages-tuned-boottime.yaml
```

4. Create a file with the following content and name it **hugepages-mcp.yaml**:

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

5. Create the machine config pool:

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

Given enough non-fragmented memory, all the nodes in the **worker-hp** machine config pool should now have 50 2Mi huge pages allocated.

```
$ oc get node <node_using_hugepages> -o jsonpath="{.status.allocatable.hugepages-2Mi}"
100Mi
```

**WARNING**

The TuneD bootloader plug-in is currently supported on Red Hat Enterprise Linux CoreOS (RHCOS) 8.x worker nodes. For Red Hat Enterprise Linux (RHEL) 7.x worker nodes, the TuneD bootloader plug-in is currently not supported.

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. LOW LATENCY TUNING

16.1. UNDERSTANDING LOW LATENCY

The emergence of Edge computing in the area of Telco / 5G plays a key role in reducing latency and congestion problems and improving application performance.

Simply put, latency determines how fast data (packets) moves from the sender to receiver and returns to the sender after processing by the receiver. Maintaining a network architecture with the lowest possible delay of latency speeds 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 numbers of 1 ms or less. This reduction in latency boosts wireless throughput by a factor of 10.

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

In an ideal world, all of those would be prioritized: in real life, some come at the expense of others. 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.

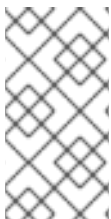
The environment in which an application is operating influences its behavior. For a typical data center with no strict latency requirements, only minimal default tuning is needed that enables CPU partitioning for some high performance workload pods. For data centers and workloads where latency is a higher 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.

In OpenShift Container Platform version 4.10 and previous versions, the Performance Addon Operator was used to implement automatic tuning to achieve low latency performance. Now this functionality is part of the Node Tuning Operator.

16.1.1. About hyperthreading for low latency and real-time applications

Hyperthreading is an Intel processor technology that allows a physical CPU processor core to function as two logical cores, executing two independent threads simultaneously. Hyperthreading allows for better system throughput for certain workload types where parallel processing is beneficial. The default OpenShift Container Platform configuration expects hyperthreading to be enabled by default.

For telecommunications applications, it is important to design your application infrastructure to minimize latency as much as possible. Hyperthreading can slow performance times and negatively affect throughput for compute intensive workloads that require low latency. Disabling hyperthreading ensures predictable performance and can decrease processing times for these workloads.



NOTE

Hyperthreading 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 hyperthreading implementation specific to that hardware. Disabling hyperthreading can increase the cost per core of the cluster.

Additional resources

- [Configuring hyperthreading for a cluster](#)

16.2. PROVISIONING REAL-TIME AND LOW LATENCY WORKLOADS

Many industries and organizations need extremely high performance computing and might require low and predictable latency, especially in the financial and telecommunications industries. For these industries, with their unique requirements, 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.

The cluster administrator can use this performance profile configuration to make these changes in a more reliable way. The administrator can specify whether to update the kernel to kernel-rt (real-time), 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.

**WARNING**

The usage of execution probes in conjunction with applications that require guaranteed CPUs can cause latency spikes. It is recommended to use other probes, such as a properly configured set of network probes, as an alternative.

**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, these functions are part of the Node Tuning Operator.

16.2.1. Known limitations for real-time

**NOTE**

In most deployments, kernel-rt is supported only on worker nodes when you use a standard cluster with three control plane nodes and three worker nodes. There are exceptions for compact and single nodes on OpenShift Container Platform deployments. For installations on a single node, kernel-rt is supported on the single control plane node.

To fully utilize the real-time mode, the containers must run with elevated privileges. See [Set capabilities for a Container](#) for information on granting privileges.

OpenShift Container Platform restricts the allowed capabilities, so you might need to create a **SecurityContext** as well.

**NOTE**

This procedure is fully supported with bare metal installations using Red Hat Enterprise Linux CoreOS (RHCOS) systems.

Establishing the right performance expectations refers to the fact that the real-time kernel is not a panacea. Its objective is consistent, low-latency determinism offering predictable response times. There is some additional kernel overhead associated with the real-time kernel. This is due primarily to handling hardware interruptions in separately scheduled threads. The increased overhead in some workloads results in some degradation in overall throughput. The exact amount of degradation is very workload dependent, ranging from 0% to 30%. However, it is the cost of determinism.

16.2.2. Provisioning a worker with real-time capabilities

1. Optional: Add a node to the OpenShift Container Platform cluster. See [Setting BIOS parameters](#).
2. Add the label **worker-rt** to the worker nodes that require the real-time capability by using the **oc** command.
3. Create a new machine config pool for real-time nodes:

```

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

```

Note that a machine config pool **worker-rt** is created for group of nodes that have the label **worker-rt**.

4. Add the node to the proper machine config pool by using node role labels.



NOTE

You must decide which nodes are configured with real-time workloads. You could configure all of the nodes in the cluster, or a subset of the nodes. The Node Tuning Operator expects all of the nodes are part of a dedicated machine config pool. If you use all of the nodes, you must point the Node Tuning Operator to the worker node role label. If you use a subset, you must group the nodes into a new machine config pool.

5. Create the **PerformanceProfile** with the proper set of housekeeping cores and **realTimeKernel: enabled: true**.
6. You must set **machineConfigPoolSelector** in **PerformanceProfile**:

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: example-performanceprofile
spec:
  ...
  realTimeKernel:
    enabled: true
  nodeSelector:
    node-role.kubernetes.io/worker-rt: ""
  machineConfigPoolSelector:
    machineconfiguration.openshift.io/role: worker-rt

```

7. Verify that a matching machine config pool exists with a label:

```
$ oc describe mcp/worker-rt
```

Example output

```
Name:      worker-rt
Namespace:
Labels:    machineconfiguration.openshift.io/role=worker-rt
```

8. OpenShift Container Platform will start configuring the nodes, which might involve multiple reboots. Wait for the nodes to settle. This can take a long time depending on the specific hardware you use, but 20 minutes per node is expected.
9. Verify everything is working as expected.

16.2.3. Verifying the real-time kernel installation

Use this command to verify that the real-time kernel is installed:

```
$ oc get node -o wide
```

Note the worker with the role **worker-rt** that contains the string **4.18.0-305.30.1.rt7.102.el8_4.x86_64 cri-o://1.24.0-99.rhaos4.10.gitc3131de.el8**:

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP
EXTERNAL-IP	OS-IMAGE			KERNEL-VERSION	
CONTAINER-RUNTIME					
rt-worker-0.example.com	Ready	worker,worker-rt	5d17h	v1.24.0	
128.66.135.107	<none>	Red Hat Enterprise Linux CoreOS	46.82.202008252340-0	(Ootpa)	
4.18.0-305.30.1.rt7.102.el8_4.x86_64		cri-o://1.24.0-99.rhaos4.10.gitc3131de.el8			
[...]					

16.2.4. Creating a workload that works in real-time

Use the following procedures for preparing a workload that will use real-time capabilities.

Procedure

1. Create a pod with a QoS class of **Guaranteed**.
2. Optional: Disable CPU load balancing for DPDK.
3. Assign a proper node selector.

When writing your applications, follow the general recommendations described in [Application tuning and deployment](#).

16.2.5. Creating a pod with a QoS class of **Guaranteed**

Keep the following in mind when you create a pod that is given a QoS class of **Guaranteed**:

- Every container in the pod must have a memory limit and a memory request, and they must be the same.
- Every container in the pod must have a CPU limit and a CPU request, and they must be the same.

The following example shows the configuration file for a pod that has one container. The container has a memory limit and a memory request, both equal to 200 MiB. The container has a CPU limit and a CPU request, both equal to 1 CPU.

```
apiVersion: v1
kind: Pod
metadata:
  name: qos-demo
  namespace: qos-example
spec:
  containers:
  - name: qos-demo-ctr
    image: <image-pull-spec>
    resources:
      limits:
        memory: "200Mi"
        cpu: "1"
      requests:
        memory: "200Mi"
        cpu: "1"
```

1. Create the pod:

```
$ oc apply -f qos-pod.yaml --namespace=qos-example
```

2. View detailed information about the pod:

```
$ oc get pod qos-demo --namespace=qos-example --output=yaml
```

Example output

```
spec:
  containers:
  ...
status:
  qosClass: Guaranteed
```



NOTE

If a container specifies its own memory limit, but does not specify a memory request, OpenShift Container Platform automatically assigns a memory request that matches the limit. Similarly, if a container specifies its own CPU limit, but does not specify a CPU request, OpenShift Container Platform automatically assigns a CPU request that matches the limit.

16.2.6. Optional: Disabling CPU load balancing for DPDK

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 pod must have the **cpu-load-balancing.crio.io: true** annotation.

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 default runtime handler except 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.

16.2.7. Assigning a proper node selector

The preferred way to assign a pod to nodes is to use the same node selector the performance profile used, as shown here:

```

apiVersion: v1
kind: Pod
metadata:
  name: example
spec:
  # ...
  nodeSelector:
    node-role.kubernetes.io/worker-rt: ""

```

For more information, see [Placing pods on specific nodes using node selectors](#) .

16.2.8. Scheduling a workload onto a worker with real-time capabilities

Use label selectors that match the nodes attached to the machine config pool that was configured for low latency by the Node Tuning Operator. For more information, see [Assigning pods to nodes](#).

16.2.9. Reducing power consumption by taking CPUs offline

You can generally anticipate telecommunication workloads. When not all of the CPU resources are required, the Node Tuning Operator allows you take unused CPUs offline to reduce power consumption by manually updating the performance profile.

To take unused CPUs offline, you must perform the following tasks:

1. Set the offline CPUs in the performance profile and save the contents of the YAML file:

Example performance profile with offlined CPUs

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  additionalKernelArgs:
    - nmi_watchdog=0
    - audit=0
    - mce=off
    - processor.max_cstate=1
    - intel_idle.max_cstate=0
    - idle=poll
  cpu:
    isolated: "2-23,26-47"
    reserved: "0,1,24,25"
    offlined: "48-59" ❶
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numa:
    topologyPolicy: single-numa-node
  realTimeKernel:
    enabled: true
```

- ❶ Optional. You can list CPUs in the **offlined** field to take the specified CPUs offline.

2. Apply the updated profile by running the following command:

```
$ oc apply -f my-performance-profile.yaml
```

16.2.10. 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.

To achieve low latency for workloads, some (but not all) pods require the CPUs they are running on to not process device interrupts. A pod annotation, **irq-load-balancing.crio.io**, is used to define whether device interrupts are processed or not. When configured, CRI-O disables device interrupts only as long as the pod is running.

16.2.10.1. Disabling CPU CFS quota

To reduce CPU throttling for individual guaranteed pods, create a pod specification with the annotation **cpu-quota.crio.io: "disable"**. This annotation disables the CPU completely fair scheduler (CFS) quota at the pod run time. The following pod specification contains this annotation:

```
apiVersion: performance.openshift.io/v2
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. Otherwise, disabling CPU CFS quota can affect the performance of other containers in the cluster.

16.2.10.2. Disabling global device interrupts handling in Node Tuning Operator

To configure Node Tuning Operator to disable global device interrupts for the isolated CPU set, set the **globallyDisableIrqLoadBalancing** field in the performance profile to **true**. When **true**, conflicting pod annotations are ignored. When **false**, IRQ loads are balanced across all CPUs.

A performance profile snippet illustrates this setting:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: manual
spec:
  globallyDisableIrqLoadBalancing: true
...
```

16.2.10.3. Disabling interrupt processing for individual pods

To disable interrupt processing for individual pods, 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>
  ...

```

16.2.11. 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.

16.2.11.1. Supported 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**.

16.2.11.1.1. 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.

16.2.11.1.2. 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**.

16.3. TUNING NODES FOR LOW LATENCY WITH THE PERFORMANCE PROFILE

The performance profile lets you control latency tuning aspects of nodes that belong to a certain machine config pool. After you specify your settings, the **PerformanceProfile** object is compiled into multiple objects that perform the actual node level tuning:

- A **MachineConfig** file that manipulates the nodes.
- A **KubeletConfig** file that configures the Topology Manager, the CPU Manager, and the OpenShift Container Platform nodes.
- The Tuned profile that configures the Node Tuning Operator.

You can use a performance profile to specify whether to update the kernel to kernel-rt, to allocate huge pages, and to partition the CPUs for performing housekeeping duties or running workloads.



NOTE

You can manually create the **PerformanceProfile** object or use the Performance Profile Creator (PPC) to generate a performance profile. See the additional resources below for more information on the PPC.

Sample performance profile

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: "5-15" 1
    reserved: "0-4" 2
  hugepages:
    defaultHugepagesSize: "1G"
  pages:
    - size: "1G"
      count: 16
      node: 0
  realTimeKernel:
    enabled: true 3
  numa: 4
    topologyPolicy: "best-effort"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: "" 5
```

- 1** Use this field to isolate specific CPUs to use with application containers for workloads.
- 2** Use this field to reserve specific CPUs to use with infra containers for housekeeping.
- 3** Use this field to install the real-time kernel on the node. Valid values are **true** or **false**. Setting the **true** value installs the real-time kernel.
- 4** Use this field to configure the topology manager policy. Valid values are **none** (default), **best-effort**, **restricted**, and **single-numa-node**. For more information, see [Topology Manager Policies](#).
- 5** Use this field to specify a node selector to apply the performance profile to specific nodes.

Additional resources

- For information on using the Performance Profile Creator (PPC) to generate a performance profile, see [Creating a performance profile](#).

16.3.1. 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-###: ###
```

16.3.2. 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
```

16.3.3. Configuring a node for IRQ dynamic load balancing

To configure a cluster node to handle IRQ dynamic load balancing, do the following:

1. Log in to the OpenShift Container Platform cluster as a user with cluster-admin privileges.
2. Set the performance profile **apiVersion** to use **performance.openshift.io/v2**.
3. Remove the **globallyDisableIrqLoadBalancing** field or set it to **false**.
4. 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, the infra containers in pods use the reserved CPUs and the application containers use the isolated CPUs.

5. Create the pod that uses exclusive CPUs, and set **irq-load-balancing.crio.io** and **cpu-quota.crio.io** annotations to **disable**. For example:

```
apiVersion: v1
kind: Pod
metadata:
  name: dynamic-irq-pod
```

```

    annotations:
      irq-load-balancing.crio.io: "disable"
      cpu-quota.crio.io: "disable"
    spec:
      containers:
      - name: dynamic-irq-pod
        image: "registry.redhat.io/openshift4/cnf-tests-rhel8:v4.11"
        command: ["sleep", "10h"]
        resources:
          requests:
            cpu: 2
            memory: "200M"
          limits:
            cpu: 2
            memory: "200M"
        nodeSelector:
          node-role.kubernetes.io/worker-cnf: ""
      runtimeClassName: performance-dynamic-irq-profile
    ...

```

6. Enter the pod **runtimeClassName** in the form `performance-<profile_name>`, where `<profile_name>` is the **name** from the **PerformanceProfile** YAML, in this example, **performance-dynamic-irq-profile**.
7. Set the node selector to target a `cnf-worker`.
8. 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
NOMINATED NODE READINESS GATES
dynamic-irq-pod 1/1   Running  0         5h33m <ip-address> <node-name> <none>
<none>

```

9. Get the CPUs that the pod configured for IRQ dynamic load balancing runs on:

```
$ oc exec -it dynamic-irq-pod -- /bin/bash -c "grep Cpus_allowed_list /proc/self/status | awk '{print $2}'"
```

Expected output

```
Cpus_allowed_list: 2-3
```

10. Ensure the node configuration is applied correctly. SSH into the node to verify the configuration.

```
$ oc debug node/<node-name>
```

Expected output

```
Starting pod/<node-name>-debug ...
To use host binaries, run `chroot /host`
```

```
Pod IP: <ip-address>
If you don't see a command prompt, try pressing enter.
```

```
sh-4.4#
```

11. Verify that you can use the node file system:

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

Expected output

```
sh-4.4#
```

12. Ensure the default system CPU affinity mask does not include the **dynamic-irq-pod** CPUs, for example, CPUs 2 and 3.

```
$ cat /proc/irq/default_smp_affinity
```

Example output

```
33
```

13. Ensure the system IRQs are not configured to run on the **dynamic-irq-pod** CPUs:

```
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
```

Some IRQ controllers do not support IRQ re-balancing and will always expose all online CPUs as the IRQ mask. These IRQ controllers effectively run on CPU 0. For more information on the host configuration, SSH into the host and run the following, replacing **<irq-num>** with the CPU number that you want to query:

```
$ cat /proc/irq/<irq-num>/effective_affinity
```

16.3.4. Configuring hyperthreading for a cluster

To configure hyperthreading 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 hyperthreading 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 hyperthreading 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 command 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.

```

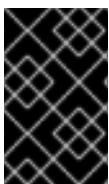
...
cpu:
  isolated: 0,4
  reserved: 1-3,5-7
...

```



NOTE

The reserved and isolated CPU pools must not overlap and together must span all available cores in the worker node.



IMPORTANT

Hyperthreading is enabled by default on most Intel processors. If you enable hyperthreading, all threads processed by a particular core must be isolated or processed on the same core.

16.3.4.1. Disabling hyperthreading for low latency applications

When configuring clusters for low latency processing, consider whether you want to disable hyperthreading before you deploy the cluster. To disable hyperthreading, do the following:

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.

16.3.5. Understanding workload hints

The following table describes how combinations of power consumption and real-time settings impact on latency.



NOTE

The following workload hints can be configured manually. You can also work with workload hints using the Performance Profile Creator. For more information about the performance profile, see the "Creating a performance profile" section.

Performance Profile creator setting	Hint	Environment	Description
Default	<div>workloadHints: highPowerConsumption: false realTime: false</div>	High throughput cluster without latency requirements	Performance achieved through CPU partitioning only.

Performance Profile creator setting	Hint	Environment	Description
Low-latency	<code>workloadHints: highPowerConsumption: false realTime: true</code>	Regional datacenters	Both energy savings and low-latency are desirable: compromise between power management, latency and throughput.
Ultra-low-latency	<code>workloadHints: highPowerConsumption: true realTime: true</code>	Far edge clusters, latency critical workloads	Optimized for absolute minimal latency and maximum determinism at the cost of increased power consumption.

Additional resources

- For information on using the Performance Profile Creator (PPC) to generate a performance profile, see [Creating a performance profile](#).

16.3.6. Configuring workload hints manually

Procedure

- Create a **PerformanceProfile** appropriate for the environment's hardware and topology as described in the table in "Understanding workload hints". Adjust the profile to match the expected workload. In this example, we tune for the lowest possible latency.
- Add the **highPowerConsumption** and **realTime** workload hints. Both are set to **true** here.

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: workload-hints
spec:
  ...
  workloadHints:
    highPowerConsumption: true 1
    realTime: true 2
```

- 1 If **highPowerConsumption** is **true**, the node is tuned for very low latency at the cost of increased power consumption.
- 2 Disables some debugging and monitoring features that can affect system latency.

16.3.7. 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 16.1. Process' CPU assignments

Process type	Details
Burstable and BestEffort 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 4.11 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.

Additional resources

- [Managing device interrupt processing for guaranteed pod isolated CPUs](#)
- [Create a pod that gets assigned a QoS class of Guaranteed](#)

16.4. REDUCING NIC QUEUES USING THE NODE TUNING OPERATOR

The Node Tuning Operator allows you to adjust the network interface controller (NIC) queue count for each network device by configuring the performance profile. Device network queues allows the distribution of packets among different physical queues and each queue gets a separate thread for packet processing.

In real-time or low latency systems, all the unnecessary interrupt request lines (IRQs) pinned to the isolated CPUs must be moved to reserved or housekeeping CPUs.

In deployments with applications that require system, OpenShift Container Platform networking or in mixed deployments with Data Plane Development Kit (DPDK) workloads, multiple queues are needed to achieve good throughput and the number of NIC queues should be adjusted or remain unchanged. For example, to achieve low latency the number of NIC queues for DPDK based workloads should be reduced to just the number of reserved or housekeeping CPUs.

Too many queues are created by default for each CPU and these do not fit into the interrupt tables for housekeeping CPUs when tuning for low latency. Reducing the number of queues makes proper tuning possible. Smaller number of queues means a smaller number of interrupts that then fit in the IRQ table.



NOTE

In earlier versions of OpenShift Container Platform, the Performance Addon Operator provided automatic, low latency performance tuning for applications. In OpenShift Container Platform 4.11, these functions are part of the Node Tuning Operator.

16.4.1. 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.

5. 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: 3-51,54-103
    reserved: 0-2,52-54
  net:
```

```

userLevelNetworking: true
nodeSelector:
  node-role.kubernetes.io/worker-cnf: ""

```

6. 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: 3-51,54-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
    devices:
      - interfaceName: "eth0"
      - interfaceName: "eth1"
      - vendorID: "0x1af4"
      - deviceID: "0x1000"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""

```

7. 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: 3-51,54-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
    devices:
      - interfaceName: "eth*"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""

```

8. 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: 3-51,54-103
    reserved: 0-2,52-54
  net:

```

```

userLevelNetworking: true
devices:
- interfaceName: "lens1"
nodeSelector:
  node-role.kubernetes.io/worker-cnf: ""

```

9. 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: 3-51,54-103
    reserved: 0-2,52-54
  net:
    userLevelNetworking: true
    devices:
    - interfaceName: "eth0"
    - vendorID: "0x1af4"
    - deviceID: "0x1000"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""

```

10. Apply the updated performance profile:

```
$ oc apply -f <your_profile_name>.yaml
```

Additional resources

- [Creating a performance profile](#) .

16.4.2. 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.

16.4.3. 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
```

16.5. PERFORMING END-TO-END TESTS FOR PLATFORM VERIFICATION

The Cloud-native Network Functions (CNF) tests image is a containerized test suite that validates features required to run CNF payloads. You can use this image to validate a CNF-enabled OpenShift cluster where all the components required for running CNF workloads are installed.

The tests run by the image are split into three different phases:

- Simple cluster validation
- Setup
- End to end tests

The validation phase checks that all the features required to be tested are deployed correctly on the cluster.

Validations include:

- Targeting a machine config pool that belong to the machines to be tested
- Enabling SCTP on the nodes
- Enabling xt_u32 kernel module via machine config
- Having the SR-IOV Operator installed
- Having the PTP Operator installed
- Enabling the **contain-mount-namespace** mode via machine config

- Using OVN-kubernetes as the cluster network provider

Latency tests, a part of the CNF-test container, also require the same validations. For more information about running a latency test, see the Running the latency tests section.

The tests need to perform an environment configuration every time they are executed. This involves items such as creating SR-IOV node policies, performance profiles, or PTP profiles. Allowing the tests to configure an already configured cluster might affect the functionality of the cluster. Also, changes to configuration items such as SR-IOV node policy might result in the environment being temporarily unavailable until the configuration change is processed.

16.5.1. Prerequisites

- The test entrypoint is **/usr/bin/test-run.sh**. It runs both a setup test set and the real conformance test suite. The minimum requirement is to provide it with a kubeconfig file and its related **\$KUBECONFIG** environment variable, mounted through a volume.
- The tests assumes that a given feature is already available on the cluster in the form of an Operator, flags enabled on the cluster, or machine configs.
- Some tests require a pre-existing machine config pool to append their changes to. This must be created on the cluster before running the tests.

The default worker pool is **worker-cnf** and can be created with the following manifest:

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

You can use the **ROLE_WORKER_CNF** variable to override the worker pool name:

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig -e
ROLE_WORKER_CNF=custom-worker-pool registry.redhat.io/openshift4/cnf-tests-
rhel8:v4.11 /usr/bin/test-run.sh
```



NOTE

Currently, not all tests run selectively on the nodes belonging to the pool.

16.5.2. Dry run

Use this command to run in dry-run mode. This is useful for checking what is in the test suite and provides output for all of the tests the image would run.

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.11 /usr/bin/test-run.sh -ginkgo.dryRun -ginkgo.v
```

16.5.3. Disconnected mode

The CNF tests image support running tests in a disconnected cluster, meaning a cluster that is not able to reach outer registries. This is done in two steps:

1. Performing the mirroring.
2. Instructing the tests to consume the images from a custom registry.

16.5.3.1. 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 images needed to run the tests to a local registry.

Run this command from an intermediate machine that has access both to the cluster and to registry.redhat.io over the internet:

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.11 /usr/bin/mirror -registry my.local.registry:5000/ | oc
image mirror -f -
```

Then, follow the instructions in the following section about overriding the registry used to fetch the images.

16.5.3.2. Instruct the tests to consume those images from a custom registry

This is done by setting the **IMAGE_REGISTRY** environment variable:

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig -e
IMAGE_REGISTRY="my.local.registry:5000/" -e CNF_TESTS_IMAGE="custom-cnf-tests-
image:latests" registry.redhat.io/openshift4/cnf-tests-rhel8:v4.11 /usr/bin/test-run.sh
```

16.5.3.3. Mirroring to the cluster internal 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:

```
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 that image stream available to all the namespaces used for tests. This is required to allow the tests namespaces to fetch the images from the **cnftests** image stream.

```
$ oc policy add-role-to-user system:image-puller system:serviceaccount:sctptest:default --
namespace=cnftests
```

```
$ 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
```

```
$ oc policy add-role-to-user system:image-puller system:serviceaccount:dpdk-testing:default
--namespace=cnftests
```

```
$ oc policy add-role-to-user system:image-puller system:serviceaccount:sriov-conformance-
testing:default --namespace=cnftests
```

```
$ oc policy add-role-to-user system:image-puller system:serviceaccount:xt-u32-testing:default
--namespace=cnftests
```

```
$ oc policy add-role-to-user system:image-puller system:serviceaccount:vrf-testing:default --
namespace=cnftests
```

```
$ oc policy add-role-to-user system:image-puller system:serviceaccount:gatekeeper-
testing:default --namespace=cnftests
```

```
$ oc policy add-role-to-user system:image-puller system:serviceaccount:ovs-qos-
testing:default --namespace=cnftests
```

5. Retrieve the docker secret name and auth token:

```
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. Write a **dockerauth.json** similar to this:

```
echo "{\"auths\": { \"$REGISTRY\": { \"auth\": $TOKEN } }}" > dockerauth.json
```

7. Do the mirroring:

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.11 /usr/bin/mirror -registry
$REGISTRY/cnftests | oc image mirror --insecure=true -a=$(pwd)/dockerauth.json -f -
```

8. Run the tests:

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig -e
IMAGE_REGISTRY=image-registry.openshift-image-registry.svc:5000/cnftests cnf-tests-
local:latest /usr/bin/test-run.sh
```

16.5.3.4. Mirroring a different set of images

Procedure

1. The **mirror** command tries to mirror the u/s 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:4.11"
  },
  {
    "registry": "public.registry.io:5000",
    "image": "imagefordpdk:4.11"
  }
]
```

2. Pass it 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.

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.11 /usr/bin/mirror --registry
"my.local.registry:5000/" --images "/kubeconfig/images.json" | oc image mirror -f -
```

16.5.4. Running in a single-node cluster

Running tests on a single-node cluster causes the following limitations to be imposed:

- Longer timeouts for certain tests, including SR-IOV and SCTP tests
- Tests requiring master and worker nodes are skipped

Longer timeouts concern SR-IOV and SCTP tests. Reconfiguration requiring node reboots cause a reboot of the entire environment, including the OpenShift control plane, and therefore takes longer to complete. All PTP tests requiring a master and worker node are skipped. No additional configuration is needed because the tests check for the number of nodes at startup and adjust test behavior accordingly.

PTP tests can run in Discovery mode. The tests look for a PTP master configured outside of the cluster.

For more information, see the Discovery mode section.

To enable Discovery mode, the tests must be instructed by setting the **DISCOVERY_MODE** environment variable as follows:

```
$ docker run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e  
DISCOVERY_MODE=true registry.redhat.io/openshift-kni/cnf-tests /usr/bin/test-run.sh
```

Required parameters

- **ROLE_WORKER_CNF=master** - Required because master is the only machine pool to which the node will belong.
- **XT_U32TEST_HAS_NON_CNF_WORKERS=false** - Required to instruct the xt_u32 negative test to skip because there are only nodes where the module is loaded.
- **SCTPTEST_HAS_NON_CNF_WORKERS=false** - Required to instruct the SCTP negative test to skip because there are only nodes where the module is loaded.

16.5.5. Impact of tests on the cluster

Depending on the feature, running the test suite could cause different impacts on the cluster. In general, only the SCTP tests do not change the cluster configuration. All of the other features have various impacts on the configuration.

16.5.5.1. SCTP

SCTP tests just run different pods on different nodes to check connectivity. The impacts on the cluster are related to running simple pods on two nodes.

16.5.5.2. XT_U32

XT_U32 tests run pods on different nodes to check iptables rule that utilize xt_u32. The impacts on the cluster are related to running simple pods on two nodes.

16.5.5.3. SR-IOV

SR-IOV tests require changes in the SR-IOV network configuration, where the tests create and destroy different types of configuration.

This might have an impact if existing SR-IOV network configurations are already installed on the cluster, because there may be conflicts depending on the priority of such configurations.

At the same time, the result of the tests might be affected by existing configurations.

16.5.5.4. PTP

PTP tests apply a PTP configuration to a set of nodes of the cluster. As with SR-IOV, this might conflict with any existing PTP configuration already in place, with unpredictable results.

16.5.5.5. Performance

Performance tests apply a performance profile to the cluster. The effect of this is changes in the node configuration, reserving CPUs, allocating memory huge pages, and setting the kernel packages to be realtime. If an existing profile named **performance** is already available on the cluster, the tests do not deploy it.

16.5.5.6. DPDK

DPDK relies on both performance and SR-IOV features, so the test suite configures both a performance profile and SR-IOV networks, so the impacts are the same as those described in SR-IOV testing and performance testing.

16.5.5.7. Container-mount-namespace

The validation test for **container-mount-namespace** mode only checks that the appropriate **MachineConfig** objects are present and active, and has no additional impact on the node.

16.5.5.8. Cleaning up

After running the test suite, all the dangling resources are cleaned up.

16.5.6. Override test image parameters

Depending on the requirements, the tests can use different images. There are two images used by the tests that can be changed using the following environment variables:

- **CNF_TESTS_IMAGE**
- **DPDK_TESTS_IMAGE**

For example, to change the **CNF_TESTS_IMAGE** with a custom registry run the following command:

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig -e
CNF_TESTS_IMAGE="custom-cnf-tests-image:latests" registry.redhat.io/openshift4/cnf-tests-
rhel8:v4.11 /usr/bin/test-run.sh
```

16.5.6.1. Ginkgo parameters

The Ginkgo BDD (Behavior-Driven Development) framework serves as the base for the test suite. This means that it accepts parameters for filtering or skipping tests.

You can use the **-ginkgo.focus** parameter to filter a set of tests:

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.11 /usr/bin/test-run.sh -
ginkgo.focus="performance|sctp"
```

You can run only the latency test using the **-ginkgo.focus** parameter.

To run only the latency test, you must provide the **-ginkgo.focus** parameter and the **PERF_TEST_PROFILE** environment variable that has the name of the **PerformanceProfile** that needs to be tested. For example:

```
$ docker run --rm -v $KUBECONFIG:/kubeconfig -e KUBECONFIG=/kubeconfig -e
LATENCY_TEST_RUN=true -e LATENCY_TEST_RUNTIME=600 -e
OSLAT_MAXIMUM_LATENCY=20 -e PERF_TEST_PROFILE=<performance_profile_name>
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.11 /usr/bin/test-run.sh -ginkgo.focus="\[performance\]\[config\]\[performance\]\ Latency\ Test"
```

**NOTE**

There is a particular test that requires both SR-IOV and SCTP. Given the selective nature of the **focus** parameter, this test is triggered by only placing the **sriov** matcher. If the tests are executed against a cluster where SR-IOV is installed but SCTP is not, adding the **-ginkgo.skip=SCTP** parameter causes the tests to skip SCTP testing.

16.5.6.2. Available features

The set of available features to filter are:

- **performance**
- **sriov**
- **ptp**
- **sctp**
- **xt_u32**
- **dpdk**
- **container-mount-namespace**

16.5.7. Discovery mode

Discovery mode allows you to validate the functionality of a cluster without altering its configuration. Existing environment configurations are used for the tests. The tests attempt to find the configuration items needed and use those items to execute the tests. If resources needed to run a specific test are not found, the test is skipped, providing an appropriate message to the user. After the tests are finished, no cleanup of the pre-configured configuration items is done, and the test environment can be immediately used for another test run.

Some configuration items are still created by the tests. These are specific items needed for a test to run; for example, a SR-IOV Network. These configuration items are created in custom namespaces and are cleaned up after the tests are executed.

An additional bonus is a reduction in test run times. As the configuration items are already there, no time is needed for environment configuration and stabilization.

To enable discovery mode, the tests must be instructed by setting the **DISCOVERY_MODE** environment variable as follows:

```
$ docker run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e
DISCOVERY_MODE=true registry.redhat.io/openshift-kni/cnf-tests /usr/bin/test-run.sh
```

16.5.7.1. Required environment configuration prerequisites

SR-IOV tests

Most SR-IOV tests require the following resources:

- **SriovNetworkNodePolicy.**

- At least one with the resource specified by **SriovNetworkNodePolicy** being allocatable; a resource count of at least 5 is considered sufficient.

Some tests have additional requirements:

- An unused device on the node with available policy resource, with link state **DOWN** and not a bridge slave.
- A **SriovNetworkNodePolicy** with a MTU value of **9000**.

DPDK tests

The DPDK related tests require:

- A performance profile.
- A SR-IOV policy.
- A node with resources available for the SR-IOV policy and available with the **PerformanceProfile** node selector.

PTP tests

- A slave **PtpConfig** (**ptp4lOpts="-s"** ,**phc2sysOpts="-a -r"**).
- A node with a label matching the slave **PtpConfig**.

SCTP tests

- **SriovNetworkNodePolicy**.
- A node matching both the **SriovNetworkNodePolicy** and a **MachineConfig** that enables SCTP.

XT_U32 tests

- A node with a machine config that enables XT_U32.

Performance Operator tests

Various tests have different requirements. Some of them are:

- A performance profile.
- A performance profile having **profile.Spec.CPU.Isolated = 1**.
- A performance profile having **profile.Spec.RealTimeKernel.Enabled == true**.
- A node with no huge pages usage.

Container-mount-namespace tests

- A node with a machine config which enables **container-mount-namespace** mode

16.5.7.2. Limiting the nodes used during tests

The nodes on which the tests are executed can be limited by specifying a **NODES_SELECTOR** environment variable. Any resources created by the test are then limited to the specified nodes.

```
$ docker run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e
NODES_SELECTOR=node-role.kubernetes.io/worker-cnf registry.redhat.io/openshift-kni/cnf-tests
/usr/bin/test-run.sh
```

16.5.7.3. Using a single performance profile

The resources needed by the DPDK tests are higher than those required by the performance test suite. To make the execution faster, the performance profile used by tests can be overridden using one that also serves the DPDK test suite.

To do this, a profile like the following one can be mounted inside the container, and the performance tests can be instructed to deploy it.

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: "4-15"
    reserved: "0-3"
  hugepages:
    defaultHugepagesSize: "1G"
  pages:
    - size: "1G"
      count: 16
      node: 0
  realTimeKernel:
    enabled: true
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```



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.

To override the performance profile used, the manifest must be mounted inside the container and the tests must be instructed by setting the **PERFORMANCE_PROFILE_MANIFEST_OVERRIDE** parameter as follows:

```
$ docker run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e
PERFORMANCE_PROFILE_MANIFEST_OVERRIDE=/kubeconfig/manifest.yaml
registry.redhat.io/openshift-kni/cnf-tests /usr/bin/test-run.sh
```

16.5.7.4. Disabling the performance profile cleanup

When not running in discovery mode, the suite cleans up all the created artifacts and configurations. This includes the performance profile.

When deleting the performance profile, the machine config pool is modified and nodes are rebooted. After a new iteration, a new profile is created. This causes long test cycles between runs.

To speed up this process, set **CLEAN_PERFORMANCE_PROFILE="false"** to instruct the tests not to clean the performance profile. In this way, the next iteration will not need to create it and wait for it to be applied.

```
$ docker run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e
CLEAN_PERFORMANCE_PROFILE="false" registry.redhat.io/openshift-kni/cnf-tests /usr/bin/test-
run.sh
```

16.5.8. Running the latency tests

If the **kubeconfig** file is in the current folder, you can run the test suite by using the following command:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e \
DISCOVERY_MODE=true registry.redhat.io/openshift4/cnf-tests-rhel8:v4.11 \
/usr/bin/test-run.sh -ginkgo.focus="[performance]\ Latency\ Test"
```

This allows the running container to use the **kubeconfig** file from inside the container.



WARNING

You must run the latency tests in Discovery mode. The latency tests can change the configuration of your cluster if you do not run in Discovery mode.

In OpenShift Container Platform 4.11, you can also run latency tests from the CNF-test container. The latency test allows you to validate node tuning for your workload.

Three tools measure the latency of the system:

- **hwlatdetect**
- **cyclicttest**
- **oslat**

Each tool has a specific use. Use the tools in sequence to achieve reliable test results.

1. The **hwlatdetect** tool measures the baseline that the bare metal hardware can achieve. Before proceeding with the next latency test, ensure that the number measured by **hwlatdetect** meets the required threshold because you cannot fix hardware latency spikes by operating system tuning.
2. The **cyclicttest** tool verifies the real-time kernel scheduler latency after **hwlatdetect** passes validation. The **cyclicttest** 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.

3. The **oslat** tool 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.

By default, the latency tests are disabled. To enable the latency test, you must add the **LATENCY_TEST_RUN** environment variable to the test invocation and set its value to **true**. For example, **LATENCY_TEST_RUN=true**.

The test introduces the following environment variables:

LATENCY_TEST_DELAY

The variable 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.

LATENCY_TEST_CPUS

The variable 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.

LATENCY_TEST_RUNTIME

The variable specifies the amount of time in seconds that the latency test must run. The default value is 300 seconds.

HWLATDETECT_MAXIMUM_LATENCY

The variable specifies the maximum acceptable hardware latency in microseconds for the workload and operating system. If you do not set the value of **HWLATDETECT_MAXIMUM_LATENCY** or **MAXIMUM_LATENCY**, the tool compares the default expected threshold (20µs) and the actual maximum latency in the tool itself. Then, the test fails or succeeds accordingly.

CYCLICTEST_MAXIMUM_LATENCY

The variable specifies the maximum latency in microseconds that all threads expect before waking up during the **cyclictest** run. If you do not set the value of **CYCLICTEST_MAXIMUM_LATENCY** or **MAXIMUM_LATENCY**, the tool skips the comparison of the expected and the actual maximum latency.

OSLAT_MAXIMUM_LATENCY

The variable specifies the maximum acceptable latency in microseconds for the **oslat** test results. If you do not set the value of **OSLAT_MAXIMUM_LATENCY** or **MAXIMUM_LATENCY**, the tool skips the comparison of the expected and the actual maximum latency.

MAXIMUM_LATENCY

This is a unified variable you can apply for all the available latency tools.



NOTE

A variable that is specific to certain tests has precedence over the unified variable.

You can use the **-ginkgo.v** flag to run the tests with verbosity.

You can use the **-ginkgo.focus** flag to run a specific test.

16.5.8.1. Running hwlatdetect

The **hwlatdetect** tool is available in the **rt-kernel** package with a regular subscription of Red Hat Enterprise Linux 8.

Prerequisites:

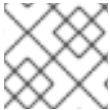
- You installed the real-time kernel
- You logged into registry.redhat.io with your Customer Portal credentials

Procedure

1. Run the following command:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e \
LATENCY_TEST_RUN=true -e DISCOVERY_MODE=true -e ROLE_WORKER_CNF=worker-cnf -e \
LATENCY_TEST_RUNTIME=600 -e MAXIMUM_LATENCY=20 registry.redhat.io/openshift4/cnf-
tests-rhel8:v4.11 \
/usr/bin/test-run.sh -ginkgo.focus="hwlatdetect"
```

The command runs the **hwlatdetect** tool for 10 minutes (600 seconds). The test runs successfully when the maximum observed latency is lower than **MAXIMUM_LATENCY** (20 μ s), and the command line displays **SUCCESS!** when this test is completed.



NOTE

For valid results, the test should run for at least 12 hours.

If the results exceed the latency threshold, the test fails and you can see the following output:

Example failure output

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e \
LATENCY_TEST_RUN=true -e DISCOVERY_MODE=true -e ROLE_WORKER_CNF=worker-cnf -e \
LATENCY_TEST_RUNTIME=10 -e MAXIMUM_LATENCY=1 registry.redhat.io/openshift4/cnf-tests-
rhel8:v4.11 \
/usr/bin/test-run.sh -ginkgo.v -ginkgo.focus="hwlatdetect" 1
```

```
running /usr/bin/validationsuite -ginkgo.v -ginkgo.focus=hwlatdetect
I0210 17:08:38.607699      7 request.go:668] Waited for 1.047200253s due to client-side throttling,
not priority and fairness, request: GET:https://api.ocp.demo.lab:6443/apis/apps.openshift.io/v1?
timeout=32s
```

Running Suite: CNF Features e2e validation

=====

Random Seed: 1644512917

Will run 0 of 48 specs

[illegible]

Ran 0 of 48 Specs in 0.001 seconds

SUCCESS! -- 0 Passed | 0 Failed | 0 Pending | 48 Skipped

PASS

Discovery mode enabled, skipping setup

```
running /usr/bin/cnftests -ginkgo.v -ginkgo.focus=hwlatdetect
```

I0210 17:08:41.179269 40 request.go:668] Waited for 1.046001096s due to client-side throttling, not priority and fairness, request: GET:https://api.ocp.demo.lab:6443/apis/storage.k8s.io/v1beta1?timeout=32s

Running Suite: CNF Features e2e integration tests

=====

Random Seed: 1644512920

Will run 1 of 151 specs

SSSSSSS

[performance] Latency Test with the hwlatdetect image
should succeed

/remote-source/app/vendor/github.com/openshift-kni/performance-addon-
operators/functests/4_latency/latency.go:221

STEP: Waiting two minutes to download the latencyTest image

STEP: Waiting another two minutes to give enough time for the cluster to move the pod to Succeeded phase

Feb 10 17:10:56.045: [INFO]: found mcd machine-config-daemon-dzpw7 for node ocp-worker-0.demo.lab

Feb 10 17:10:56.259: [INFO]: found mcd machine-config-daemon-dzpw7 for node ocp-worker-0.demo.lab

Feb 10 17:11:56.825: [ERROR]: timed out waiting for the condition

• Failure [193.903 seconds]

[performance] Latency Test

/remote-source/app/vendor/github.com/openshift-kni/performance-addon-
operators/functests/4_latency/latency.go:60

with the hwlatdetect image

/remote-source/app/vendor/github.com/openshift-kni/performance-addon-
operators/functests/4_latency/latency.go:213

should succeed [It]

/remote-source/app/vendor/github.com/openshift-kni/performance-addon-
operators/functests/4_latency/latency.go:221

Log file created at: 2022/02/10 17:08:45

Running on machine: hwlatdetect-cd8b6

Binary: Built with gc go1.16.6 for linux/amd64

Log line format: [IWEF]mmdd hh:mm:ss.uuuuuu threadid file:line] msg

I0210 17:08:45.716288 1 node.go:37] Environment information: /proc/cmdline: BOOT_IMAGE=(hd0,gpt3)/ostree/rhcos-

56fabcb39a679b757ebae30e5f01b2ebd38e9fde9ecae91c41be41d3e89b37f8/vmlinuz-4.18.0-

305.34.2.rt7.107.el8_4.x86_64 random.trust_cpu=on console=tty0 console=ttyS0,115200n8

ignition.platform.id=qemu

ostree=/ostree/boot.0/rhcos/56fabcb39a679b757ebae30e5f01b2ebd38e9fde9ecae91c41be41d3e89b3

7f8/0 root=UUID=56731f4f-f558-46a3-85d3-d1b579683385 rw rootflags=prjquota skew_tick=1

nohz=on rcu_nocbs=3-5 tuned.non_isolcpus=fffffc7 intel_pstate=disable nosoftlockup

tsc=nowatchdog intel_iommu=on iommu=pt isolcpus=managed_irq,3-5

systemd.cpu_affinity=0,1,2,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31

+ +

I0210 17:08:45.716782 1 node.go:44] Environment information: kernel version 4.18.0-305.34.2.rt7.107.el8_4.x86_64

I0210 17:08:45.716861 1 main.go:50] running the hwlatdetect command with arguments
[/usr/bin/hwlatdetect --threshold 1 --hardlimit 1 --duration 10 --window 10000000us --width 950000us]

F0210 17:08:56.815204 1 main.go:53] failed to run hwlatdetect command; out: hwlatdetect:
test duration 10 seconds

detector: tracer

parameters:

Latency threshold: 1us **2**

Sample window: 10000000us

Sample width: 950000us

Non-sampling period: 9050000us

Output File: None

Starting test

test finished

Max Latency: 24us **3**

Samples recorded: 1

Samples exceeding threshold: 1

ts: 1644512927.163556381, inner:20, outer:24

```
; err: exit status 1
```

```
goroutine 1 [running]:
```

```
k8s.io/klog.stacks(0xc00010001, 0xc00012e000, 0x25b, 0x2710)
```

```
/remote-source/app/vendor/k8s.io/klog/klog.go:875 +0xb9
```

```
k8s.io/klog.(*loggingT).output(0x5bed00, 0xc000000003, 0xc0000121c0, 0x53ea81, 0x7, 0x35,
```

0x0)

```
/remote-source/app/vendor/k8s.io/klog/klog.go:829 +0x1b0
```

```
k8s.io/klog.(*loggingT).printf(0x5bed00, 0x3, 0x5082da, 0x33, 0xc000113f58, 0x2, 0x2)
```

```
/remote-source/app/vendor/k8s.io/klog/klog.go:707 +0x153
```

k8s.io/klog.Fatalf(...)

/remote-source/app/vendor/k8s.io/klog/klog.go:1276

```
main.main()
```

```
/remote-source/app/cnf-tests/pod-utils/hwlatdetect-runner/main.go:53 +0x897
```

```
goroutine 6 [chan receive]:
```

```
k8s.io/klog.(*loggingT).flushDaemon(0x5bed00)
```

```
/remote-source/app/vendor/k8s.io/klog/klog.go:1010 +0x8b
```

created by k8s.io/klog.init.0

```

/remote-source/app/vendor/k8s.io/klog/klog.go:411 +0xd8

```

```
goroutine 7 [chan receive]:
```

```
k8s.io/klog/v2.(*loggingT).flushDaemon(0x5bede0)
```

```

/remote-source/app/vendor/k8s.io/klog/v2/klog.go:1169 +0x8b

```

created by k8s.io/klog/v2.init.0

```
/remote-source/app/vendor/k8s.io/klog/v2/klog.go:420 +0xdf
```

Unexpected error:

```
<*errors.errorString | 0xc000418ed0>: {
```

s: "timed out waiting for the condition",

}

timed out waiting for the condition

occurred

```
/remote-source/app/vendor/github.com/openshift-kni/performance-addon-
```

operators/functests/4_latency/latency.go:433

[illegible]

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-kni/performance-addon-operators/functests/4_latency/latency.go:433
```

Ran 1 of 151 Specs in 222.254 seconds

```
FAIL! -- 0 Passed | 1 Failed | 0 Pending | 150 Skipped
```

```
--- FAIL: TestTest (222.45s)
```

```
FAIL
```

- 1 The **podman** arguments you provided.
- 2 You can configure the latency threshold by using the **MAXIMUM_LATENCY** or the **HWLATDETECT_MAXIMUM_LATENCY** environment variables.
- 3 The maximum latency value measured during the test.

16.5.8.1.1. Capturing the 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 BIOS settings



WARNING

Before proceeding with the next latency test, ensure that the number measured by **hwlatdetect** meets the required threshold. Fixing latencies introduced by hardware might require you to contact the support of your system vendor.

16.5.8.2. Running **cyclicttest**

The **cyclicttest** tool measures the real-time kernel scheduler latency on the specified CPUs.

Prerequisites

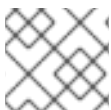
- You logged into registry.redhat.io with your Customer Portal credentials
- You installed the real-time kernel
- You applied the performance profile by using the Node Tuning Operator

Procedure

To perform the **cyclicttest**, run the following command:

```
$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e \
LATENCY_TEST_RUN=true -e DISCOVERY_MODE=true -e ROLE_WORKER_CNF=worker-cnf -e \
LATENCY_TEST_CPUS=10 -e LATENCY_TEST_RUNTIME=600 -e MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.11 /usr/bin/test-run.sh -ginkgo.focus="cyclicttest"
```

The command runs the **cyclicttest** tool for 10 minutes (600 seconds). The test runs successfully when the maximum observed latency is lower than **MAXIMUM_LATENCY** (20 μ s), and the command line displays **SUCCESS!** when this test is completed.



NOTE

For valid results, the test should run for at least 12 hours.

If the results exceed the latency threshold, the test fails and you can see the following output:

Example failure output

```
$ podman run -v $(pwd)/:kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e \
PERF_TEST_PROFILE=<performance_profile_name> -e ROLE_WORKER_CNF=worker-cnf -e \
LATENCY_TEST_RUN=true -e LATENCY_TEST_RUNTIME=600 -e MAXIMUM_LATENCY=20 -e \
LATENCY_TEST_CPUS=10 -e DISCOVERY_MODE=true \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.11 usr/bin/test-run.sh \
-ginkgo.v -ginkgo.focus="cyclictest" 1
```

[illegible]

```
[performance] Latency Test with the cyclictest image
should succeed
/go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-kni/performance-
addon-operators/functests/4_latency/latency.go:200
STEP: Waiting two minutes to download the latencyTest image
STEP: Waiting another two minutes to give enough time for the cluster to move the pod to Succeeded
phase
Aug 11 15:03:06.826: [INFO]: found mcd machine-config-daemon-wf4w8 for node
cnfdc8.clus2.t5g.lab.eng.bos.redhat.com
```

- Failure [22.527 seconds]
[performance] Latency Test
/go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-kni/performance-addon-operators/func-tests/4_latency/latency.go:84
with the cyclictest image
/go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-kni/performance-addon-operators/func-tests/4_latency/latency.go:188
should succeed [It]
/go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-kni/performance-addon-operators/func-tests/4_latency/latency.go:200

The current latency 17 is bigger than the expected one 20 **2**
 Expected
 <bool>: false
 to be true

```

/go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-kni/performance-
addon-operators/func-tests/4 latency/latency.go:219

```

Log file created at: 2021/08/11 15:02:51
Running on machine: cyclictest-**knk7d**

```

Binary: Built with gc go1.16.6 for linux/amd64
Log line format: [IWEF]mmdd hh:mm:ss.uuuuuu threadid file:line] msg
I0811 15:02:51.092254      1 node.go:37] Environment information: /proc/cmdline: BOOT_IMAGE=
(hd0,gpt3)/ostree/rhcos-
612d89f4519a53ad0b1a132f4add78372661bfb3994f5fe115654971aa58a543/vmlinuz-4.18.0-
305.10.2.rt7.83.el8_4.x86_64 ip=dhcp random.trust_cpu=on console=tty0 console=ttyS0,115200n8
ostree=/ostree/boot.1/rhcos/612d89f4519a53ad0b1a132f4add78372661bfb3994f5fe115654971aa58a5
43/0 ignition.platform.id=openstack root=UUID=5a4ddf16-9372-44d9-ac4e-3ee329e16ab3 rw
rootflags=prjquota skew_tick=1 nohz=on rcu_nocbs=1-3
tuned.non_isolcpus=000000ff,ffffff,ffffff,ffffff1 intel_pstate=disable nosoftlockup tsc=nowatchdog
intel_iommu=on iommu=pt isolcpus=managed_irq,1-3
systemd.cpu_affinity=0,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,6
4,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,
97,98,99,100,101,102,103 default_hugepagesz=1G hugepagesz=2M hugepages=128
nmi_watchdog=0 audit=0 mce=off processor.max_cstate=1 idle=poll intel_idle.max_cstate=0
I0811 15:02:51.092427      1 node.go:44] Environment information: kernel version 4.18.0-
305.10.2.rt7.83.el8_4.x86_64
I0811 15:02:51.092450      1 main.go:48] running the cyclicttest command with arguments \
[-D 600 -95 1 -t 10 -a 2,4,6,8,10,54,56,58,60,62 -h 30 -i 1000 --quiet] 3
I0811 15:03:06.147253      1 main.go:54] succeeded to run the cyclicttest command: #
/dev/cpu_dma_latency set to 0us
# Histogram
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000
000001 000000 005561 027778 037704 011987 000000 120755 238981 081847 300186
000002 587440 581106 564207 554323 577416 590635 474442 357940 513895 296033
000003 011751 011441 006449 006761 008409 007904 002893 002066 003349 003089
000004 000527 001079 000914 000712 001451 001120 000779 000283 000350 000251

More histogram entries ...
# Min Latencies: 00002 00001 00001 00001 00001 00002 00001 00001 00001 00001
# Avg Latencies: 00002 00002 00002 00001 00002 00002 00001 00001 00001 00001
# Max Latencies: 00018 00465 00361 00395 00208 00301 02052 00289 00327 00114 4
# Histogram Overflows: 00000 00220 00159 00128 00202 00017 00069 00059 00045 00120
# Histogram Overflow at cycle number:
# Thread 0:
# Thread 1: 01142 01439 05305 ... # 00190 others
# Thread 2: 20895 21351 30624 ... # 00129 others
# Thread 3: 01143 17921 18334 ... # 00098 others
# Thread 4: 30499 30622 31566 ... # 00172 others
# Thread 5: 145221 170910 171888 ...
# Thread 6: 01684 26291 30623 ...# 00039 others
# Thread 7: 28983 92112 167011 ... 00029 others
# Thread 8: 45766 56169 56171 ...# 00015 others
# Thread 9: 02974 08094 13214 ... # 00090 others

```

- 1 The **podman** arguments you provided.
- 2 You can see the measured latency and the configured latency.
- 3 The arguments for the **cyclicttest** command.
- 4 The maximum latencies measured on each thread.

16.5.8.2.1. Capturing the results

The same output can indicate different results for different workloads. For example, spikes up to 18µs is acceptable for 4G DU workloads but not for 5G DU workloads. Spikes above 20µs are not acceptable in any case.

Example of good results

```
running cmd: cyclictst -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 00002
# Avg Latencies: 00002 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: cyclictst -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 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 00002
# Avg Latencies: 00002 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

```

16.5.8.3. Running oslat

Prerequisites

- You logged into registry.redhat.io with your Customer Portal credentials
- You applied the performance profile by using the Node Tuning Operator

Procedure

- To perform the **oslat**, run the following command:

```

$ podman run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e \
LATENCY_TEST_RUN=true -e DISCOVERY_MODE=true -e ROLE_WORKER_CNF=worker-cnf -e \
LATENCY_TEST_CPUS=7 -e LATENCY_TEST_RUNTIME=600 -e MAXIMUM_LATENCY=20 \
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.11 /usr/bin/test-run.sh -ginkgo.focus="oslat"

```

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 μ s), and the command line displays **SUCCESS!** when this test is completed.

If the results exceed the latency threshold, the test fails and you can see the following output:

to be true

/go/src/github.com/openshift-kni/cnf-features-deploy/vendor/github.com/openshift-kni/performance-addon-operators/functests/4_latency/latency.go:168

Log file created at: 2021/08/29 13:25:21

Running on machine: oslat-57c2g

Binary: Built with gc go1.16.6 for linux/amd64

Log line format: [IWEF]mmdd hh:mm:ss.uuuuuu threadid file:line] msg

I0829 13:25:21.569182 1 node.go:37] Environment information: /proc/cmdline: BOOT_IMAGE=(hd0,gpt3)/ostree/rhcos-

612d89f4519a53ad0b1a132f4add78372661bfb3994f5fe115654971aa58a543/vmlinuz-4.18.0-

305.10.2.rt7.83.el8_4.x86_64 ip=dhcp random.trust_cpu=on console=tty0 console=ttyS0,115200n8

ostree=/ostree/boot.0/rhcos/612d89f4519a53ad0b1a132f4add78372661bfb3994f5fe115654971aa58a5

43/0 ignition.platform.id=openstack root=UUID=5a4ddf16-9372-44d9-ac4e-3ee329e16ab3 rw

rootflags=prjquota skew_tick=1 nohz=on rcu_nocbs=1-3

tuned.non_isolcpus=000000ff,ffffff,ffffff,ffffff1 intel_pstate=disable nosoftlockup tsc=nowatchdog

intel_iommu=on iommu=pt isolcpus=managed_irq,1-3

systemd.cpu_affinity=0,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31

,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,6

4,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,

97,98,99,100,101,102,103 default_hugepagesz=1G hugepagesz=2M hugepages=128

nmi_watchdog=0 audit=0 mce=off processor.max_cstate=1 idle=poll intel_idle.max_cstate=0

I0829 13:25:21.569345 1 node.go:44] Environment information: kernel version 4.18.0-

305.10.2.rt7.83.el8_4.x86_64

I0829 13:25:21.569367 1 main.go:53] Running the oslat command with arguments \

[--duration 600 --rtprio 1 --cpu-list 4,6,52,54,56,58 --cpu-main-thread 2] **3**

I0829 13:35:22.632263 1 main.go:59] Succeeded to run the oslat command: oslat V 2.00

Total runtime: 600 seconds

Thread priority: SCHED_FIFO:1

CPU list: 4,6,52,54,56,58

CPU for main thread: 2

Workload: no

Workload mem: 0 (KiB)

Preheat cores: 6

Pre-heat for 1 seconds...

Test starts...

Test completed.

Core: 4 6 52 54 56 58

CPU Freq: 2096 2096 2096 2096 2096 2096 (Mhz)

001 (us): 19390720316 19141129810 20265099129 20280959461 19391991159 19119877333

002 (us): 5304 5249 5777 5947 6829 4971

003 (us): 28 14 434 47 208 21

004 (us): 1388 853 123568 152817 5576 0

005 (us): 207850 223544 103827 91812 227236 231563

006 (us): 60770 122038 277581 323120 122633 122357

007 (us): 280023 223992 63016 25896 214194 218395

008 (us): 40604 25152 24368 4264 24440 25115

009 (us): 6858 3065 5815 810 3286 2116

010 (us): 1947 936 1452 151 474 361

...

Minimum: 1 1 1 1 1 1 (us)

Average: 1.000 1.000 1.000 1.000 1.000 1.000 (us)

```
Maximum: 37 38 49 28 28 19 (us) 4
Max-Min: 36 37 48 27 27 18 (us)
Duration: 599.667 599.667 599.667 599.667 599.667 599.667 (sec)
```

- 1 3 The list of CPUs running the **oslat** command. The **LATENCY_TEST_CPUS** variable provides seven CPUs. You can only see six CPUs in total because one runs the **oslat** tool.
- 2 You can see the measured latency and the configured latency.
- 4 The maximum latency values in microseconds that each CPU measures.

16.5.9. Troubleshooting

The cluster must be reached from within the container. You can verify this by running:

```
$ docker run -v $(pwd)/:/kubeconfig -e KUBECONFIG=/kubeconfig/kubeconfig
registry.redhat.io/openshift-kni/cnf-tests oc get nodes
```

If this does not work, it could be caused by spanning across DNS, MTU size, or firewall issues.

16.5.10. Test reports

CNF end-to-end tests produce two outputs: a JUnit test output and a test failure report.

16.5.10.1. JUnit test output

A JUnit-compliant XML is produced by passing the **--junit** parameter together with the path where the report is dumped:

```
$ docker run -v $(pwd)/:/kubeconfig -v $(pwd)/junitdest:/path/to/junit -e
KUBECONFIG=/kubeconfig/kubeconfig registry.redhat.io/openshift4/cnf-tests-rhel8:v4.11
/usr/bin/test-run.sh --junit /path/to/junit
```

16.5.10.2. Test failure report

A report with information about the cluster state and resources for troubleshooting can be produced by passing the **--report** parameter with the path where the report is dumped:

```
$ docker run -v $(pwd)/:/kubeconfig -v $(pwd)/reportdest:/path/to/report -e
KUBECONFIG=/kubeconfig/kubeconfig registry.redhat.io/openshift4/cnf-tests-rhel8:v4.11
/usr/bin/test-run.sh --report /path/to/report
```

16.5.10.3. A note on podman

When executing podman as non root and non privileged, mounting paths can fail with "permission denied" errors. To make it work, append **:Z** to the volumes creation; for example, **-v \$(pwd)/:/kubeconfig:Z** to allow podman to do the proper SELinux relabeling.

16.5.10.4. Running on OpenShift Container Platform 4.4

With the exception of the following, the CNF end-to-end tests are compatible with OpenShift Container Platform 4.4:

```
[test_id:28466][crit:high][vendor:cnf-qe@redhat.com][level:acceptance] Should contain configuration injected through openshift-node-performance profile
[test_id:28467][crit:high][vendor:cnf-qe@redhat.com][level:acceptance] Should contain configuration injected through the openshift-node-performance profile
```

You can skip these tests by adding the **-ginkgo.skip** “28466|28467” parameter.

16.5.10.5. Using a single performance profile

The DPDK tests require more resources than what is required by the performance test suite. To make the execution faster, you can override the performance profile used by the tests using a profile that also serves the DPDK test suite.

To do this, use a profile like the following one that can be mounted inside the container, and the performance tests can be instructed to deploy it.

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: "5-15"
    reserved: "0-4"
  hugepages:
    defaultHugepagesSize: "1G"
  pages:
    - size: "1G"
      count: 16
      node: 0
  realTimeKernel:
    enabled: true
  numa:
    topologyPolicy: "best-effort"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```



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.

To override the performance profile, the manifest must be mounted inside the container and the tests must be instructed by setting the **PERFORMANCE_PROFILE_MANIFEST_OVERRIDE**:

```
$ docker run -v $(pwd)/:/kubeconfig:Z -e KUBECONFIG=/kubeconfig/kubeconfig -e
PERFORMANCE_PROFILE_MANIFEST_OVERRIDE=/kubeconfig/manifest.yaml
registry.redhat.io/openshift4/cnf-tests-rhel8:v4.11 /usr/bin/test-run.sh
```

16.6. 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.

16.6.1. 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		
DEGRADEDMACHINECOUNT	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

```

16.7. 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.

16.7.1. 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.

16.7.2. About collecting 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.

To collect debugging information with **must-gather**, you must specify the Performance Addon Operator **must-gather** image:

```
--image=registry.redhat.io/openshift4/performance-addon-operator-must-gather-rhel8:v4.11.
```



NOTE

In earlier versions of OpenShift Container Platform, the Performance Addon Operator provided automatic, low latency performance tuning for applications. In OpenShift Container Platform 4.11, these functions are part of the Node Tuning Operator. However, you must still use the **performance-addon-operator-must-gather** image when running the **must-gather** command.

16.7.3. Gathering data about specific features

You can gather debugging information about specific features by using the **oc adm must-gather** CLI command with the **--image** or **--image-stream** argument. The **must-gather** tool supports multiple images, so you can gather data about more than one feature by running a single command.



NOTE

To collect the default **must-gather** data in addition to specific feature data, add the **--image-stream=openshift/must-gather** argument.



NOTE

In earlier versions of OpenShift Container Platform, the Performance Addon Operator provided automatic, low latency performance tuning for applications. In OpenShift Container Platform 4.11, these functions are part of the Node Tuning Operator. However, you must still use the **performance-addon-operator-must-gather** image when running the **must-gather** command.

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. Run the **oc adm must-gather** command with one or more **--image** or **--image-stream** arguments. For example, the following command gathers both the default cluster data and information specific to the Node Tuning Operator:

```
$ oc adm must-gather \
  --image-stream=openshift/must-gather \ 1
```

```
--image=registry.redhat.io/openshift4/performance-addon-operator-must-gather-rhel8:v4.11
```

2**1**

The default OpenShift Container Platform **must-gather** image.

2

The **must-gather** image for low latency tuning diagnostics.

3. Create a compressed file from the **must-gather** directory that was created in your working directory. For example, on a computer that uses a Linux operating system, run the following command:

```
$ tar cvaf must-gather.tar.gz must-gather.local.5421342344627712289/ 1
```

1

Replace **must-gather-local.5421342344627712289/** with the actual directory name.

4. Attach the compressed file to your support case on the [Red Hat Customer Portal](#).

Additional resources

- For more information about MachineConfig and KubeletConfig, see [Managing nodes](#).
- For more information about the Node Tuning Operator, see [Using the Node Tuning Operator](#).
- For more information about the PerformanceProfile, see [Configuring huge pages](#).
- For more information about consuming huge pages from your containers, see [How huge pages are consumed by apps](#).

CHAPTER 17. IMPROVING CLUSTER STABILITY IN HIGH LATENCY ENVIRONMENTS USING WORKER LATENCY PROFILES

All nodes send heartbeats to the Kubernetes Controller Manager Operator (kube controller) in the OpenShift Container Platform cluster every 10 seconds, by default. If the cluster does not receive heartbeats from a node, OpenShift Container Platform responds using several default mechanisms.

For example, if the Kubernetes Controller Manager Operator loses contact with a node after a configured period:

1. The node controller on the control plane updates the node health to **Unhealthy** and marks the node **Ready** condition as **Unknown**.
2. In response, the scheduler stops scheduling pods to that node.
3. The on-premise node 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 Operator might not receive an update from a healthy node due to network latency. The Kubernetes Controller Manager Operator would then evict 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 Operator wait for status updates before taking action. These adjustments help to ensure that your cluster runs properly in the event that network latency between the control plane and the worker nodes is not optimal.

These worker latency profiles are three sets of parameters that are pre-defined with carefully tuned values that let you control the reaction of the cluster to latency issues without needing to determine the best values manually.

These worker latency profiles are three sets of parameters that are pre-defined with carefully tuned values that control the reaction of the cluster to latency issues without your needing to determine 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.

17.1. UNDERSTANDING WORKER LATENCY PROFILES

Worker latency profiles are multiple sets of carefully-tuned values for the **node-status-update-frequency**, **node-monitor-grace-period**, **default-not-ready-toleration-seconds** and **default-unreachable-toleration-seconds** parameters. These parameters let you control the reaction of the cluster to latency issues without needing to determine the best values manually.

All worker latency profiles configure the following parameters:

- **node-status-update-frequency**. Specifies the amount of time in seconds that a kubelet updates its status to the Kubernetes Controller Manager Operator.
- **node-monitor-grace-period**. Specifies the amount of time in seconds that the Kubernetes Controller Manager Operator 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 Kubernetes Controller Manager 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 Kubernetes Controller Manager Operator waits before evicting pods from that node.



IMPORTANT

Manually modifying the **node-monitor-grace-period** parameter is not supported.

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

Default worker latency profile

With the **Default** profile, each kubelet reports its node status to the Kubelet Controller Manager Operator (kube controller) every 10 seconds. The Kubelet Controller Manager Operator checks the kubelet for a status every 5 seconds.

The Kubernetes Controller Manager Operator waits 40 seconds for a status update before considering that node unhealthy. It marks the node with the **node.kubernetes.io/not-ready** or **node.kubernetes.io/unreachable** taint and evicts the pods on that node. If a pod on that node has the **NoExecute** toleration, the pod gets evicted in 300 seconds. If the pod has the **tolerationSeconds** parameter, the eviction waits for the period specified by that parameter.

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

Medium worker latency profile

Use the **MediumUpdateAverageReaction** profile if the network latency is slightly higher than usual. The **MediumUpdateAverageReaction** profile reduces the frequency of kubelet updates to 20 seconds and changes the period that the Kubernetes Controller Manager Operator 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 Operator waits for 2 minutes to consider a node unhealthy. In another minute, the eviction process starts.

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

Low worker latency profile

Use the **LowUpdateSlowReaction** profile if the network latency is extremely high.

The **LowUpdateSlowReaction** profile reduces the frequency of kubelet updates to 1 minute and changes the period that the Kubernetes Controller Manager Operator 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.

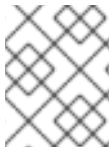
The Kubernetes Controller Manager Operator waits for 5 minutes to consider a node unhealthy. In another minute, the eviction process starts.

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

17.2. USING WORKER LATENCY PROFILES

To implement 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 the 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.

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.

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 to use the low worker latency policy.

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

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

CHAPTER 18. TOPOLOGY AWARE LIFECYCLE MANAGER FOR CLUSTER UPDATES

You can use the Topology Aware Lifecycle Manager (TALM) to manage the software lifecycle of multiple single-node OpenShift clusters. TALM uses Red Hat Advanced Cluster Management (RHACM) policies to perform changes on the target clusters.



IMPORTANT

Topology Aware Lifecycle Manager is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see <https://access.redhat.com/support/offerings/techpreview/>.

18.1. ABOUT THE TOPOLOGY AWARE LIFECYCLE MANAGER CONFIGURATION

The Topology Aware Lifecycle Manager (TALM) manages the deployment of Red Hat Advanced Cluster Management (RHACM) policies for one or more OpenShift Container Platform clusters. Using TALM in a large network of clusters allows the phased rollout of policies to the clusters in limited batches. This helps to minimize possible service disruptions when updating. With TALM, you can control the following actions:

- The timing of the update
- The number of RHACM-managed clusters
- The subset of managed clusters to apply the policies to
- The update order of the clusters
- The set of policies remediated to the cluster
- The order of policies remediated to the cluster

TALM supports the orchestration of the OpenShift Container Platform y-stream and z-stream updates, and day-two operations on y-streams and z-streams.

18.2. ABOUT MANAGED POLICIES USED WITH TOPOLOGY AWARE LIFECYCLE MANAGER

The Topology Aware Lifecycle Manager (TALM) uses RHACM policies for cluster updates.

TALM can be used to manage the rollout of any policy CR where the **remediationAction** field is set to **inform**. Supported use cases include the following:

- Manual user creation of policy CRs

- Automatically generated policies from the **PolicyGenTemplate** custom resource definition (CRD)

For policies that update an Operator subscription with manual approval, TALM provides additional functionality that approves the installation of the updated Operator.

For more information about managed policies, see [Policy Overview](#) in the RHACM documentation.

For more information about the **PolicyGenTemplate** CRD, see the "About the PolicyGenTemplate" section in "Deploying distributed units at scale in a disconnected environment".

18.3. INSTALLING THE TOPOLOGY AWARE LIFECYCLE MANAGER BY USING THE WEB CONSOLE

You can use the OpenShift Container Platform web console to install the Topology Aware Lifecycle Manager.

Prerequisites

- Install the latest version of the RHACM Operator.
- Set up a hub cluster with disconnected registry.
- Log in as a user with **cluster-admin** privileges.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators → OperatorHub**.
2. Search for the **Topology Aware Lifecycle Manager** from the list of available Operators, and then click **Install**.
3. Keep the default selection of **Installation mode** ["All namespaces on the cluster (default)"] and **Installed Namespace** ("openshift-operators") to ensure that the Operator is installed properly.
4. Click **Install**.

Verification

To confirm that the installation is successful:

1. Navigate to the **Operators → Installed Operators** page.
2. Check that the Operator is installed in the **All Namespaces** namespace and its status is **Succeeded**.

If the Operator is not installed successfully:

1. Navigate to the **Operators → Installed Operators** page and inspect the **Status** column for any errors or failures.
2. Navigate to the **Workloads → Pods** page and check the logs in any containers in the **cluster-group-upgrades-controller-manager** pod that are reporting issues.

18.4. INSTALLING THE TOPOLOGY AWARE LIFECYCLE MANAGER BY USING THE CLI

You can use the OpenShift CLI (**oc**) to install the Topology Aware Lifecycle Manager (TALM).

Prerequisites

- Install the OpenShift CLI (**oc**).
- Install the latest version of the RHACM Operator.
- Set up a hub cluster with disconnected registry.
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Create a **Subscription** CR:
 - a. Define the **Subscription** CR and save the YAML file, for example, **talm-subscription.yaml**:

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

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

```
$ oc create -f talm-subscription.yaml
```

Verification

1. Verify that the installation succeeded by inspecting the CSV resource:

```
$ oc get csv -n openshift-operators
```

Example output

NAME	DISPLAY	VERSION
REPLACES	PHASE	
topology-aware-lifecycle-manager.4.11.x	Topology Aware Lifecycle Manager	4.11.x
Succeeded		

2. Verify that the TALM is up and running:

```
$ oc get deploy -n openshift-operators
```

Example output

NAMESPACE	NAME	READY	UP-TO-
DATE AVAILABLE AGE			
openshift-operators	cluster-group-upgrades-controller-manager		1/1
1 1 14s			

18.5. ABOUT THE CLUSTERGROUPUPGRADE CR

The Topology Aware Lifecycle Manager (TALM) builds the remediation plan from the **ClusterGroupUpgrade** CR for a group of clusters. You can define the following specifications in a **ClusterGroupUpgrade** CR:

- Clusters in the group
- Blocking **ClusterGroupUpgrade** CRs
- Applicable list of managed policies
- Number of concurrent updates
- Applicable canary updates
- Actions to perform before and after the update
- Update timing

As TALM works through remediation of the policies to the specified clusters, the **ClusterGroupUpgrade** CR can have the following states:

- **UpgradeNotStarted**
- **UpgradeCannotStart**
- **UpgradeNotComplete**
- **UpgradeTimedOut**
- **UpgradeCompleted**
- **PrecachingRequired**



NOTE

After TALM completes a cluster update, the cluster does not update again under the control of the same **ClusterGroupUpgrade** CR. You must create a new **ClusterGroupUpgrade** CR in the following cases:

- When you need to update the cluster again
- When the cluster changes to non-compliant with the **inform** policy after being updated

18.5.1. The UpgradeNotStarted state

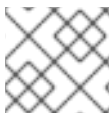
The initial state of the **ClusterGroupUpgrade** CR is **UpgradeNotStarted**.

TALM builds a remediation plan based on the following fields:

- The **clusterSelector** field specifies the labels of the clusters that you want to update.
- The **clusters** field specifies a list of clusters to update.
- The **canaries** field specifies the clusters for canary updates.
- The **maxConcurrency** field specifies the number of clusters to update in a batch.

You can use the **clusters** and the **clusterSelector** fields together to create a combined list of clusters.

The remediation plan starts with the clusters listed in the **canaries** field. Each canary cluster forms a single-cluster batch.



NOTE

Any failures during the update of a canary cluster stops the update process.

The **ClusterGroupUpgrade** CR transitions to the **UpgradeNotCompleted** state after the remediation plan is successfully created and after the **enable** field is set to **true**. At this point, TALM starts to update the non-compliant clusters with the specified managed policies.



NOTE

You can only make changes to the **spec** fields if the **ClusterGroupUpgrade** CR is either in the **UpgradeNotStarted** or the **UpgradeCannotStart** state.

Sample ClusterGroupUpgrade CR in the UpgradeNotStarted state

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-upgrade-complete
  namespace: default
spec:
  clusters: ❶
  - spoke1
  enable: false
  managedPolicies: ❷
  - policy1-common-cluster-version-policy
  - policy2-common-nto-sub-policy
  remediationStrategy: ❸
  canaries: ❹
  - spoke1
  maxConcurrency: 1 ❺
  timeout: 240
status: ❻
conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
```

```

type: Ready
copiedPolicies:
- cgu-upgrade-complete-policy1-common-cluster-version-policy
- cgu-upgrade-complete-policy2-common-nto-sub-policy
managedPoliciesForUpgrade:
- name: policy1-common-cluster-version-policy
  namespace: default
- name: policy2-common-nto-sub-policy
  namespace: default
placementBindings:
- cgu-upgrade-complete-policy1-common-cluster-version-policy
- cgu-upgrade-complete-policy2-common-nto-sub-policy
placementRules:
- cgu-upgrade-complete-policy1-common-cluster-version-policy
- cgu-upgrade-complete-policy2-common-nto-sub-policy
remediationPlan:
- - spoke1

```

- 1 Defines the list of clusters to update.
- 2 Lists the user-defined set of policies to remediate.
- 3 Defines the specifics of the cluster updates.
- 4 Defines the clusters for canary updates.
- 5 Defines the maximum number of concurrent updates in a batch. The number of remediation batches is the number of canary clusters, plus the number of clusters, except the canary clusters, divided by the **maxConcurrency** value. The clusters that are already compliant with all the managed policies are excluded from the remediation plan.
- 6 Displays information about the status of the updates.

18.5.2. The UpgradeCannotStart state

In the **UpgradeCannotStart** state, the update cannot start because of the following reasons:

- Blocking CRs are missing from the system
- Blocking CRs have not yet finished

18.5.3. The UpgradeNotCompleted state

In the **UpgradeNotCompleted** state, TALM enforces the policies following the remediation plan defined in the **UpgradeNotStarted** state.

Enforcing the policies for subsequent batches starts immediately after all the clusters of the current batch are compliant with all the managed policies. If the batch times out, TALM moves on to the next batch. The timeout value of a batch is the **spec.timeout** field divided by the number of batches in the remediation plan.



NOTE

The managed policies apply in the order that they are listed in the **managedPolicies** field in the **ClusterGroupUpgrade** CR. One managed policy is applied to the specified clusters at a time. After the specified clusters comply with the current policy, the next managed policy is applied to the next non-compliant cluster.

Sample ClusterGroupUpgrade CR in the UpgradeNotCompleted state

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-upgrade-complete
  namespace: default
spec:
  clusters:
    - spoke1
  enable: true ❶
  managedPolicies:
    - policy1-common-cluster-version-policy
    - policy2-common-nto-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status: ❷
  conditions:
    - message: The ClusterGroupUpgrade CR has upgrade policies that are still non compliant
      reason: UpgradeNotCompleted
      status: "False"
      type: Ready
  copiedPolicies:
    - cgu-upgrade-complete-policy1-common-cluster-version-policy
    - cgu-upgrade-complete-policy2-common-nto-sub-policy
  managedPoliciesForUpgrade:
    - name: policy1-common-cluster-version-policy
      namespace: default
    - name: policy2-common-nto-sub-policy
      namespace: default
  placementBindings:
    - cgu-upgrade-complete-policy1-common-cluster-version-policy
    - cgu-upgrade-complete-policy2-common-nto-sub-policy
  placementRules:
    - cgu-upgrade-complete-policy1-common-cluster-version-policy
    - cgu-upgrade-complete-policy2-common-nto-sub-policy
  remediationPlan:
    - - spoke1
  status:
    currentBatch: 1
    remediationPlanForBatch: ❸
    spoke1: 0
```

❶ The update starts when the value of the **spec.enable** field is **true**.

❷ The **status** fields change accordingly when the update begins.

- 3 Lists the clusters in the batch and the index of the policy that is being currently applied to each cluster. The index of the policies starts with **0** and the index follows the order of the

18.5.4. The UpgradeTimedOut state

In the **UpgradeTimedOut** state, TALM checks every hour if all the policies for the **ClusterGroupUpgrade** CR are compliant. The checks continue until the **ClusterGroupUpgrade** CR is deleted or the updates are completed. The periodic checks allow the updates to complete if they get prolonged due to network, CPU, or other issues.

TALM transitions to the **UpgradeTimedOut** state in two cases:

- When the current batch contains canary updates and the cluster in the batch does not comply with all the managed policies within the batch timeout.
- When the clusters do not comply with the managed policies within the **timeout** value specified in the **remediationStrategy** field.

If the policies are compliant, TALM transitions to the **UpgradeCompleted** state.

18.5.5. The UpgradeCompleted state

In the **UpgradeCompleted** state, the cluster updates are complete.

Sample ClusterGroupUpgrade CR in the UpgradeCompleted state

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-upgrade-complete
  namespace: default
spec:
  actions:
    afterCompletion:
      deleteObjects: true 1
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-nto-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
  status: 2
  conditions:
  - message: The ClusterGroupUpgrade CR has all clusters compliant with all the managed policies
    reason: UpgradeCompleted
    status: "True"
    type: Ready
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-nto-sub-policy
```

```

namespace: default
remediationPlan:
- - spoke1
status:
  remediationPlanForBatch:
    spoke1: -2 3

```

- 1 The value of **spec.action.afterCompletion.deleteObjects** field is **true** by default. After the update is completed, TALM deletes the underlying RHACM objects that were created during the update. This option is to prevent the RHACM hub from continuously checking for compliance after a successful update.
- 2 The **status** fields show that the updates completed successfully.
- 3 Displays that all the policies are applied to the cluster.

<discreet><title>The PrecachingRequired state</title>

In the **PrecachingRequired** state, the clusters need to have images pre-cached before the update can start. For more information about pre-caching, see the "Using the container image pre-cache feature" section.

</discreet>

18.5.6. Blocking ClusterGroupUpgrade CRs

You can create multiple **ClusterGroupUpgrade** CRs and control their order of application.

For example, if you create **ClusterGroupUpgrade** CR C that blocks the start of **ClusterGroupUpgrade** CR A, then **ClusterGroupUpgrade** CR A cannot start until the status of **ClusterGroupUpgrade** CR C becomes **UpgradeComplete**.

One **ClusterGroupUpgrade** CR can have multiple blocking CRs. In this case, all the blocking CRs must complete before the upgrade for the current CR can start.

Prerequisites

- Install the Topology Aware Lifecycle Manager (TALM).
- Provision one or more managed clusters.
- Log in as a user with **cluster-admin** privileges.
- Create RHACM policies in the hub cluster.

Procedure

1. Save the content of the **ClusterGroupUpgrade** CRs in the **cgu-a.yaml**, **cgu-b.yaml**, and **cgu-c.yaml** files.

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-a
  namespace: default
spec:

```

```

blockingCRs: ❶
- name: cgu-c
  namespace: default
clusters:
- spoke1
- spoke2
- spoke3
enable: false
managedPolicies:
- policy1-common-cluster-version-policy
- policy2-common-pao-sub-policy
- policy3-common-ntp-sub-policy
remediationStrategy:
  canaries:
  - spoke1
  maxConcurrency: 2
  timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ntp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ntp-sub-policy
    namespace: default
  placementBindings:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ntp-sub-policy
  placementRules:
  - cgu-a-policy1-common-cluster-version-policy
  - cgu-a-policy2-common-pao-sub-policy
  - cgu-a-policy3-common-ntp-sub-policy
  remediationPlan:
  - - spoke1
    - - spoke2

```

- ❶ Defines the blocking CRs. The **cgu-a** update cannot start until **cgu-c** is complete.

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-b
  namespace: default
spec:
  blockingCRs: ❶

```



```

- name: cgu-a
  namespace: default
clusters:
- spoke4
- spoke5
enable: false
managedPolicies:
- policy1-common-cluster-version-policy
- policy2-common-pao-sub-policy
- policy3-common-ntp-sub-policy
- policy4-common-sriov-sub-policy
remediationStrategy:
  maxConcurrency: 1
  timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR is not enabled
    reason: UpgradeNotStarted
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ntp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy2-common-pao-sub-policy
    namespace: default
  - name: policy3-common-ntp-sub-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ntp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-b-policy1-common-cluster-version-policy
  - cgu-b-policy2-common-pao-sub-policy
  - cgu-b-policy3-common-ntp-sub-policy
  - cgu-b-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke4
  - - spoke5
  status: {}

```

- 1 The **cgu-b** update cannot start until **cgu-a** is complete.

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-c

```

```

namespace: default
spec: ❶
  clusters:
  - spoke6
  enable: false
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ntp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
  status:
    conditions:
    - message: The ClusterGroupUpgrade CR is not enabled
      reason: UpgradeNotStarted
      status: "False"
      type: Ready
    copiedPolicies:
    - cgu-c-policy1-common-cluster-version-policy
    - cgu-c-policy4-common-sriov-sub-policy
    managedPoliciesCompliantBeforeUpgrade:
    - policy2-common-pao-sub-policy
    - policy3-common-ntp-sub-policy
    managedPoliciesForUpgrade:
    - name: policy1-common-cluster-version-policy
      namespace: default
    - name: policy4-common-sriov-sub-policy
      namespace: default
    placementBindings:
    - cgu-c-policy1-common-cluster-version-policy
    - cgu-c-policy4-common-sriov-sub-policy
    placementRules:
    - cgu-c-policy1-common-cluster-version-policy
    - cgu-c-policy4-common-sriov-sub-policy
    remediationPlan:
    - - spoke6
  status: {}

```

- ❶ The **cgu-c** update does not have any blocking CRs. TALM starts the **cgu-c** update when the **enable** field is set to **true**.

2. Create the **ClusterGroupUpgrade** CRs by running the following command for each relevant CR:

```
$ oc apply -f <name>.yaml
```

3. Start the update process by running the following command for each relevant CR:

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/<name> \
--type merge -p '{"spec":{"enable":true}}'
```

The following examples show **ClusterGroupUpgrade** CRs where the **enable** field is set to **true**:

Example for cgu-a with blocking CRs

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-a
  namespace: default
spec:
  blockingCRs:
    - name: cgu-c
      namespace: default
  clusters:
    - spoke1
    - spoke2
    - spoke3
  enable: true
  managedPolicies:
    - policy1-common-cluster-version-policy
    - policy2-common-pao-sub-policy
    - policy3-common-ntp-sub-policy
  remediationStrategy:
    canaries:
      - spoke1
    maxConcurrency: 2
    timeout: 240
status:
  conditions:
    - message: 'The ClusterGroupUpgrade CR is blocked by other CRs that have not yet
      completed: [cgu-c]' 1
      reason: UpgradeCannotStart
      status: "False"
      type: Ready
  copiedPolicies:
    - cgu-a-policy1-common-cluster-version-policy
    - cgu-a-policy2-common-pao-sub-policy
    - cgu-a-policy3-common-ntp-sub-policy
  managedPoliciesForUpgrade:
    - name: policy1-common-cluster-version-policy
      namespace: default
    - name: policy2-common-pao-sub-policy
      namespace: default
    - name: policy3-common-ntp-sub-policy
      namespace: default
  placementBindings:
    - cgu-a-policy1-common-cluster-version-policy
    - cgu-a-policy2-common-pao-sub-policy
    - cgu-a-policy3-common-ntp-sub-policy
  placementRules:
    - cgu-a-policy1-common-cluster-version-policy
    - cgu-a-policy2-common-pao-sub-policy
    - cgu-a-policy3-common-ntp-sub-policy
  remediationPlan:
    - - spoke1
    - - spoke2
  status: {}

```

- 1 Shows the list of blocking CRs.

Example for **cgu-b** with blocking CRs

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-b
  namespace: default
spec:
  blockingCRs:
    - name: cgu-a
      namespace: default
  clusters:
    - spoke4
    - spoke5
  enable: true
  managedPolicies:
    - policy1-common-cluster-version-policy
    - policy2-common-pao-sub-policy
    - policy3-common-ntp-sub-policy
    - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
  conditions:
    - message: 'The ClusterGroupUpgrade CR is blocked by other CRs that have not yet
      completed: [cgu-a]' 1
      reason: UpgradeCannotStart
      status: "False"
      type: Ready
  copiedPolicies:
    - cgu-b-policy1-common-cluster-version-policy
    - cgu-b-policy2-common-pao-sub-policy
    - cgu-b-policy3-common-ntp-sub-policy
    - cgu-b-policy4-common-sriov-sub-policy
  managedPoliciesForUpgrade:
    - name: policy1-common-cluster-version-policy
      namespace: default
    - name: policy2-common-pao-sub-policy
      namespace: default
    - name: policy3-common-ntp-sub-policy
      namespace: default
    - name: policy4-common-sriov-sub-policy
      namespace: default
  placementBindings:
    - cgu-b-policy1-common-cluster-version-policy
    - cgu-b-policy2-common-pao-sub-policy
    - cgu-b-policy3-common-ntp-sub-policy
    - cgu-b-policy4-common-sriov-sub-policy
  placementRules:
    - cgu-b-policy1-common-cluster-version-policy
    - cgu-b-policy2-common-pao-sub-policy
    - cgu-b-policy3-common-ntp-sub-policy

```

```
- cgu-b-policy4-common-sriov-sub-policy
remediationPlan:
- - spoke4
- - spoke5
status: {}
```

- 1 Shows the list of blocking CRs.

Example for cgu-c with blocking CRs

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-c
  namespace: default
spec:
  clusters:
  - spoke6
  enable: true
  managedPolicies:
  - policy1-common-cluster-version-policy
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  - policy4-common-sriov-sub-policy
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240
status:
  conditions:
  - message: The ClusterGroupUpgrade CR has upgrade policies that are still non compliant
    1
    reason: UpgradeNotCompleted
    status: "False"
    type: Ready
  copiedPolicies:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  managedPoliciesCompliantBeforeUpgrade:
  - policy2-common-pao-sub-policy
  - policy3-common-ptp-sub-policy
  managedPoliciesForUpgrade:
  - name: policy1-common-cluster-version-policy
    namespace: default
  - name: policy4-common-sriov-sub-policy
    namespace: default
  placementBindings:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  placementRules:
  - cgu-c-policy1-common-cluster-version-policy
  - cgu-c-policy4-common-sriov-sub-policy
  remediationPlan:
  - - spoke6
  status:
```

```
currentBatch: 1
remediationPlanForBatch:
  spoke6: 0
```

- 1 The **cgu-c** update does not have any blocking CRs.

18.6. UPDATE POLICIES ON MANAGED CLUSTERS

The Topology Aware Lifecycle Manager (TALM) remediates a set of **inform** policies for the clusters specified in the **ClusterGroupUpgrade** CR. TALM remediates **inform** policies by making **enforce** copies of the managed RHACM policies. Each copied policy has its own corresponding RHACM placement rule and RHACM placement binding.

One by one, TALM adds each cluster from the current batch to the placement rule that corresponds with the applicable managed policy. If a cluster is already compliant with a policy, TALM skips applying that policy on the compliant cluster. TALM then moves on to applying the next policy to the non-compliant cluster. After TALM completes the updates in a batch, all clusters are removed from the placement rules associated with the copied policies. Then, the update of the next batch starts.

If a spoke cluster does not report any compliant state to RHACM, the managed policies on the hub cluster can be missing status information that TALM needs. TALM handles these cases in the following ways:

- If a policy's **status.compliant** field is missing, TALM ignores the policy and adds a log entry. Then, TALM continues looking at the policy's **status.status** field.
- If a policy's **status.status** is missing, TALM produces an error.
- If a cluster's compliance status is missing in the policy's **status.status** field, TALM considers that cluster to be non-compliant with that policy.

For more information about RHACM policies, see [Policy overview](#).

Additional resources

For more information about **PolicyGenTemplate** CRD, see [About the PolicyGenTemplate](#).

18.6.1. Applying update policies to managed clusters

You can update your managed clusters by applying your policies.

Prerequisites

- Install the Topology Aware Lifecycle Manager (TALM).
- Provision one or more managed clusters.
- Log in as a user with **cluster-admin** privileges.
- Create RHACM policies in the hub cluster.

Procedure

1. Save the contents of the **ClusterGroupUpgrade** CR in the **cgu-1.yaml** file.

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-1
  namespace: default
spec:
  managedPolicies: ❶
    - policy1-common-cluster-version-policy
    - policy2-common-nto-sub-policy
    - policy3-common-ptp-sub-policy
    - policy4-common-sriov-sub-policy
  enable: false
  clusters: ❷
    - spoke1
    - spoke2
    - spoke5
    - spoke6
  remediationStrategy:
    maxConcurrency: 2 ❸
    timeout: 240 ❹

```

- ❶ The name of the policies to apply.
- ❷ The list of clusters to update.
- ❸ The **maxConcurrency** field signifies the number of clusters updated at the same time.
- ❹ The update timeout in minutes.

2. Create the **ClusterGroupUpgrade** CR by running the following command:

```
$ oc create -f cgu-1.yaml
```

- a. Check if the **ClusterGroupUpgrade** CR was created in the hub cluster by running the following command:

```
$ oc get cgu --all-namespaces
```

Example output

```

NAMESPACE  NAME    AGE
default    cgu-1   8m55s

```

- b. Check the status of the update by running the following command:

```
$ oc get cgu -n default cgu-1 -ojsonpath='{.status}' | jq
```

Example output

```

{
  "computedMaxConcurrency": 2,
  "conditions": [

```

```

    {
      "lastTransitionTime": "2022-02-25T15:34:07Z",
      "message": "The ClusterGroupUpgrade CR is not enabled", 1
      "reason": "UpgradeNotStarted",
      "status": "False",
      "type": "Ready"
    }
  ],
  "copiedPolicies": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-nto-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "managedPoliciesContent": {
    "policy1-common-cluster-version-policy": "null",
    "policy2-common-nto-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"node-tuning-operator\",\"namespace\":\"openshift-cluster-node-tuning-operator\"}]",
    "policy3-common-ptp-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"ptp-operator-subscription\",\"namespace\":\"openshift-ptp\"}]",
    "policy4-common-sriov-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"sriov-network-operator-subscription\",\"namespace\":\"openshift-sriov-network-operator\"}]"
  },
  "managedPoliciesForUpgrade": [
    {
      "name": "policy1-common-cluster-version-policy",
      "namespace": "default"
    },
    {
      "name": "policy2-common-nto-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy3-common-ptp-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy4-common-sriov-sub-policy",
      "namespace": "default"
    }
  ],
  "managedPoliciesNs": {
    "policy1-common-cluster-version-policy": "default",
    "policy2-common-nto-sub-policy": "default",
    "policy3-common-ptp-sub-policy": "default",
    "policy4-common-sriov-sub-policy": "default"
  },
  "placementBindings": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-nto-sub-policy",
    "cgu-policy3-common-ptp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "placementRules": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-nto-sub-policy",

```



```

    "cgu-policy3-common-ntp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "precaching": {
    "spec": {}
  },
  "remediationPlan": [
    [
      "spoke1",
      "spoke2"
    ],
    [
      "spoke5",
      "spoke6"
    ]
  ],
  "status": {}
}

```

- 1 The **spec.enable** field in the **ClusterGroupUpgrade** CR is set to **false**.

- c. Check the status of the policies by running the following command:

```
$ oc get policies -A
```

Example output

NAMESPACE	NAME	COMPLIANCE STATE	AGE	REMEDIATION ACTION
default	cgu-policy1-common-cluster-version-policy	enforce	17m	
default	cgu-policy2-common-ntp-sub-policy	enforce	17m	
default	cgu-policy3-common-ntp-sub-policy	enforce	17m	
default	cgu-policy4-common-sriov-sub-policy	enforce	17m	
default	policy1-common-cluster-version-policy	inform	15h	NonCompliant
default	policy2-common-ntp-sub-policy	inform	15h	NonCompliant
default	policy3-common-ntp-sub-policy	inform	18m	NonCompliant
default	policy4-common-sriov-sub-policy	inform	18m	NonCompliant

- 1 The **spec.remediationAction** field of policies currently applied on the clusters is set to **enforce**. The managed policies in **inform** mode from the **ClusterGroupUpgrade** CR remain in **inform** mode during the update.

3. Change the value of the **spec.enable** field to **true** by running the following command:

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-1 \
--patch '{"spec":{"enable":true}}' --type=merge
```

Verification

1. Check the status of the update again by running the following command:

```
$ oc get cgu -n default cgu-1 -ojsonpath='{.status}' | jq
```

Example output

```
{
  "computedMaxConcurrency": 2,
  "conditions": [ 1
    {
      "lastTransitionTime": "2022-02-25T15:34:07Z",
      "message": "The ClusterGroupUpgrade CR has upgrade policies that are still non
compliant",
      "reason": "UpgradeNotCompleted",
      "status": "False",
      "type": "Ready"
    }
  ],
  "copiedPolicies": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-nto-sub-policy",
    "cgu-policy3-common-ntp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "managedPoliciesContent": {
    "policy1-common-cluster-version-policy": "null",
    "policy2-common-nto-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"node-tuning-
operator\",\"namespace\":\"openshift-cluster-node-tuning-operator\"}]",
    "policy3-common-ntp-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"ntp-operator-
subscription\",\"namespace\":\"openshift-ntp\"}]",
    "policy4-common-sriov-sub-policy": "[{\"kind\":\"Subscription\",\"name\":\"sriov-network-
operator-subscription\",\"namespace\":\"openshift-sriov-network-operator\"}]"
  },
  "managedPoliciesForUpgrade": [
    {
      "name": "policy1-common-cluster-version-policy",
      "namespace": "default"
    },
    {
      "name": "policy2-common-nto-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy3-common-ntp-sub-policy",
      "namespace": "default"
    },
    {
      "name": "policy4-common-sriov-sub-policy",
      "namespace": "default"
    }
  ]
}
```

```

    }
  ],
  "managedPoliciesNs": {
    "policy1-common-cluster-version-policy": "default",
    "policy2-common-nto-sub-policy": "default",
    "policy3-common-ntp-sub-policy": "default",
    "policy4-common-sriov-sub-policy": "default"
  },
  "placementBindings": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-nto-sub-policy",
    "cgu-policy3-common-ntp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "placementRules": [
    "cgu-policy1-common-cluster-version-policy",
    "cgu-policy2-common-nto-sub-policy",
    "cgu-policy3-common-ntp-sub-policy",
    "cgu-policy4-common-sriov-sub-policy"
  ],
  "precaching": {
    "spec": {}
  },
  "remediationPlan": [
    [
      "spoke1",
      "spoke2"
    ],
    [
      "spoke5",
      "spoke6"
    ]
  ],
  "status": {
    "currentBatch": 1,
    "currentBatchStartedAt": "2022-02-25T15:54:16Z",
    "remediationPlanForBatch": {
      "spoke1": 0,
      "spoke2": 1
    },
    "startedAt": "2022-02-25T15:54:16Z"
  }
}

```

1 Reflects the update progress of the current batch. Run this command again to receive updated information about the progress.

2. If the policies include Operator subscriptions, you can check the installation progress directly on the single-node cluster.
 - a. Export the **KUBECONFIG** file of the single-node cluster you want to check the installation progress for by running the following command:

```
$ export KUBECONFIG=<cluster_kubeconfig_absolute_path>
```

- b. Check all the subscriptions present on the single-node cluster and look for the one in the policy you are trying to install through the **ClusterGroupUpgrade** CR by running the following command:

```
$ oc get subs -A | grep -i <subscription_name>
```

Example output for cluster-logging policy

NAMESPACE	NAME	PACKAGE	SOURCE
CHANNEL			
openshift-logging	cluster-logging	cluster-logging	redhat-
operators stable			

3. If one of the managed policies includes a **ClusterVersion** CR, check the status of platform updates in the current batch by running the following command against the spoke cluster:

```
$ oc get clusterversion
```

Example output

NAME	VERSION	AVAILABLE	PROGRESSING	SINCE	STATUS
version	4.9.5	True	True	43s	Working towards 4.9.7: 71 of 735 done (9% complete)

4. Check the Operator subscription by running the following command:

```
$ oc get subs -n <operator-namespace> <operator-subscription> -ojsonpath="{.status}"
```

5. Check the install plans present on the single-node cluster that is associated with the desired subscription by running the following command:

```
$ oc get installplan -n <subscription_namespace>
```

Example output for cluster-logging Operator

NAMESPACE	NAME	CSV	APPROVAL
APPROVED			
openshift-logging	install-6khtw	cluster-logging.5.3.3-4	Manual true

- 1 The install plans have their **Approval** field set to **Manual** and their **Approved** field changes from **true** to **false** after TALM approves the install plan.

6. Check if the cluster service version for the Operator of the policy that the **ClusterGroupUpgrade** is installing reached the **Succeeded** phase by running the following command:

```
$ oc get csv -n <operator_namespace>
```

Example output for OpenShift Logging Operator

NAME	DISPLAY	VERSION	REPLACES	PHASE
cluster-logging.5.4.2	Red Hat OpenShift Logging	5.4.2		Succeeded

18.7. CREATING A BACKUP OF CLUSTER RESOURCES BEFORE UPGRADE

For single-node OpenShift, the Topology Aware Lifecycle Manager (TALM) can create a backup of a deployment before an upgrade. If the upgrade fails, you can recover the previous version and restore a cluster to a working state without requiring a reprovision of applications.

The container image backup starts when the **backup** field is set to **true** in the **ClusterGroupUpgrade** CR.

The backup process can be in the following statuses:

BackupStatePreparingToStart

The first reconciliation pass is in progress. The TALM deletes any spoke backup namespace and hub view resources that have been created in a failed upgrade attempt.

BackupStateStarting

The backup prerequisites and backup job are being created.

BackupStateActive

The backup is in progress.

BackupStateSucceeded

The backup has succeeded.

BackupStateTimeout

Artifact backup has been partially done.

BackupStateError

The backup has ended with a non-zero exit code.



NOTE

If the backup fails and enters the **BackupStateTimeout** or **BackupStateError** state, the cluster upgrade does not proceed.

18.7.1. Creating a ClusterGroupUpgrade CR with backup

For single-node OpenShift, you can create a backup of a deployment before an upgrade. If the upgrade fails you can use the **upgrade-recovery.sh** script generated by Topology Aware Lifecycle Manager (TALM) to return the system to its preupgrade state. The backup consists of the following items:

Cluster backup

A snapshot of **etcd** and static pod manifests.

Content backup

Backups of folders, for example, **/etc**, **/usr/local**, **/var/lib/kubelet**.

Changed files backup

Any files managed by **machine-config** that have been changed.

Deployment

A pinned **ostree** deployment.

Images (Optional)

Any container images that are in use.

Prerequisites

- Install the Topology Aware Lifecycle Manager (TALM).
- Provision one or more managed clusters.
- Log in as a user with **cluster-admin** privileges.
- Install Red Hat Advanced Cluster Management (RHACM).

NOTE

It is highly recommended that you create a recovery partition. The following is an example **SiteConfig** custom resource (CR) for a recovery partition of 50 GB:

```
nodes:
  - hostname: "snode.sno-worker-0.e2e.bos.redhat.com"
    role: "master"
    rootDeviceHints:
      hctl: "0:2:0:0"
      deviceName: /dev/sda
  .....
  .....
  #Disk /dev/sda: 893.3 GiB, 959119884288 bytes, 1873281024 sectors
  diskPartition:
    - device: /dev/sda
  partitions:
    - mount_point: /var/recovery
      size: 51200
      start: 800000
```

Procedure

1. Save the contents of the **ClusterGroupUpgrade** CR with the **backup** field set to **true** in the **clustergroupupgrades-group-du.yaml** file:

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: du-upgrade-4918
  namespace: ztp-group-du-sno
spec:
  preCaching: true
  backup: true
  clusters:
    - cnfdb1
    - cnfdb2
  enable: false
  managedPolicies:
    - du-upgrade-platform-upgrade
```

```
remediationStrategy:
  maxConcurrency: 2
  timeout: 240
```

2. To start the update, apply the **ClusterGroupUpgrade** CR by running the following command:

```
$ oc apply -f clustergroupupgrades-group-du.yaml
```

Verification

- Check the status of the upgrade in the hub cluster by running the following command:

```
$ oc get cgu -n ztp-group-du-sno du-upgrade-4918 -o jsonpath='{.status}'
```

Example output

```
{
  "backup": {
    "clusters": [
      "cnfdb2",
      "cnfdb1"
    ],
    "status": {
      "cnfdb1": "Succeeded",
      "cnfdb2": "Succeeded"
    }
  },
  "computedMaxConcurrency": 1,
  "conditions": [
    {
      "lastTransitionTime": "2022-04-05T10:37:19Z",
      "message": "Backup is completed",
      "reason": "BackupCompleted",
      "status": "True",
      "type": "BackupDone"
    }
  ],
  "precaching": {
    "spec": {}
  },
  "status": {}
}
```

18.7.2. Recovering a cluster after a failed upgrade

If an upgrade of a cluster fails, you can manually log in to the cluster and use the backup to return the cluster to its preupgrade state. There are two stages:

Rollback

If the attempted upgrade included a change to the platform OS deployment, you must roll back to the previous version before running the recovery script.

Recovery

The recovery shuts down containers and uses files from the backup partition to relaunch containers and restore clusters.

Prerequisites

- Install the Topology Aware Lifecycle Manager (TALM).
- Provision one or more managed clusters.
- Install Red Hat Advanced Cluster Management (RHACM).
- Log in as a user with **cluster-admin** privileges.
- Run an upgrade that is configured for backup.

Procedure

1. Delete the previously created **ClusterGroupUpgrade** custom resource (CR) by running the following command:

```
$ oc delete cgu/du-upgrade-4918 -n ztp-group-du-sno
```

2. Log in to the cluster that you want to recover.
3. Check the status of the platform OS deployment by running the following command:

```
$ oc ostree admin status
```

Example outputs

```
[root@lab-test-spoke2-node-0 core]# ostree admin status
* rhcos c038a8f08458bbbed83a77ece033ad3c55597e3f64edad66ea12fda18cbdceaf9.0
  Version: 49.84.202202230006-0
  Pinned: yes ❶
  origin refs spec:
c038a8f08458bbbed83a77ece033ad3c55597e3f64edad66ea12fda18cbdceaf9
```

- ❶ The current deployment is pinned. A platform OS deployment rollback is not necessary.

```
[root@lab-test-spoke2-node-0 core]# ostree admin status
* rhcos f750ff26f2d5550930ccbe17af61af47daafc8018cd9944f2a3a6269af26b0fa.0
  Version: 410.84.202204050541-0
  origin refs spec: f750ff26f2d5550930ccbe17af61af47daafc8018cd9944f2a3a6269af26b0fa
rhcos ad8f159f9dc4ea7e773fd9604c9a16be0fe9b266ae800ac8470f63abc39b52ca.0
(rollback) ❶
  Version: 410.84.202203290245-0
  Pinned: yes ❷
  origin refs spec:
ad8f159f9dc4ea7e773fd9604c9a16be0fe9b266ae800ac8470f63abc39b52ca
```

- ❶ This platform OS deployment is marked for rollback.
- ❷ The previous deployment is pinned and can be rolled back.

4. To trigger a rollback of the platform OS deployment, run the following command:

```
■
```



```
$ rpm-ostree rollback -r
```

- The first phase of the recovery shuts down containers and restores files from the backup partition to the targeted directories. To begin the recovery, run the following command:

```
$ /var/recovery/upgrade-recovery.sh
```

- When prompted, reboot the cluster by running the following command:

```
$ systemctl reboot
```

- After the reboot, restart the recovery by running the following command:

```
$ /var/recovery/upgrade-recovery.sh --resume
```



NOTE

If the recovery utility fails, you can retry with the **--restart** option:

```
$ /var/recovery/upgrade-recovery.sh --restart
```

Verification

- To check the status of the recovery run the following command:

```
$ oc get clusterversion,nodes,clusteroperator
```

Example output

```
NAME                                VERSION  AVAILABLE  PROGRESSING  SINCE
STATUS
clusterversion.config.openshift.io/version  4.9.23  True       False       86d  Cluster
version is 4.9.23 ❶

NAME                                STATUS  ROLES      AGE  VERSION
node/lab-test-spoke1-node-0  Ready  master,worker  86d  v1.22.3+b93fd35 ❷

NAME                                VERSION  AVAILABLE
PROGRESSING  DEGRADED  SINCE  MESSAGE
clusteroperator.config.openshift.io/authentication  4.9.23  True    False
False  2d7h ❸
clusteroperator.config.openshift.io/baremetal      4.9.23  True    False
False  86d
```

❶ The cluster version is available and has the correct version.

❷ The node status is **Ready**.

- 3 The **ClusterOperator** object's availability is **True**.

18.8. USING THE CONTAINER IMAGE PRE-CACHE FEATURE

Clusters might have limited bandwidth to access the container image registry, which can cause a timeout before the updates are completed.



NOTE

The time of the update is not set by TALM. You can apply the **ClusterGroupUpgrade** CR at the beginning of the update by manual application or by external automation.

The container image pre-caching starts when the **preCaching** field is set to **true** in the **ClusterGroupUpgrade** CR. After a successful pre-caching process, you can start remediating policies. The remediation actions start when the **enable** field is set to **true**.

The pre-caching process can be in the following statuses:

PrecacheNotStarted

This is the initial state all clusters are automatically assigned to on the first reconciliation pass of the **ClusterGroupUpgrade** CR.

In this state, TALM deletes any pre-caching namespace and hub view resources of spoke clusters that remain from previous incomplete updates. TALM then creates a new **ManagedClusterView** resource for the spoke pre-caching namespace to verify its deletion in the **PrecachePreparing** state.

PrecachePreparing

Cleaning up any remaining resources from previous incomplete updates is in progress.

PrecacheStarting

Pre-caching job prerequisites and the job are created.

PrecacheActive

The job is in "Active" state.

PrecacheSucceeded

The pre-cache job has succeeded.

PrecacheTimeout

The artifact pre-caching has been partially done.

PrecacheUnrecoverableError

The job ends with a non-zero exit code.

18.8.1. Creating a ClusterGroupUpgrade CR with pre-caching

The pre-cache feature allows the required container images to be present on the spoke cluster before the update starts.

Prerequisites

- Install the Topology Aware Lifecycle Manager (TALM).
- Provision one or more managed clusters.

- Log in as a user with **cluster-admin** privileges.

Procedure

1. Save the contents of the **ClusterGroupUpgrade** CR with the **preCaching** field set to **true** in the **clustergroupupgrades-group-du.yaml** file:

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: du-upgrade-4918
  namespace: ztp-group-du-sno
spec:
  preCaching: true 1
  clusters:
    - cnfdb1
    - cnfdb2
  enable: false
  managedPolicies:
    - du-upgrade-platform-upgrade
  remediationStrategy:
    maxConcurrency: 2
    timeout: 240
```

- 1 The **preCaching** field is set to **true**, which enables TALM to pull the container images before starting the update.

2. When you want to start the update, apply the **ClusterGroupUpgrade** CR by running the following command:

```
$ oc apply -f clustergroupupgrades-group-du.yaml
```

Verification

1. Check if the **ClusterGroupUpgrade** CR exists in the hub cluster by running the following command:

```
$ oc get cgu -A
```

Example output

```
NAMESPACE      NAME                AGE
ztp-group-du-sno du-upgrade-4918    10s 1
```

- 1 The CR is created.

2. Check the status of the pre-caching task by running the following command:

```
$ oc get cgu -n ztp-group-du-sno du-upgrade-4918 -o jsonpath='{.status}'
```

Example output

```

{
  "conditions": [
    {
      "lastTransitionTime": "2022-01-27T19:07:24Z",
      "message": "Precaching is not completed (required)", 1
      "reason": "PrecachingRequired",
      "status": "False",
      "type": "Ready"
    },
    {
      "lastTransitionTime": "2022-01-27T19:07:24Z",
      "message": "Precaching is required and not done",
      "reason": "PrecachingNotDone",
      "status": "False",
      "type": "PrecachingDone"
    },
    {
      "lastTransitionTime": "2022-01-27T19:07:34Z",
      "message": "Pre-caching spec is valid and consistent",
      "reason": "PrecacheSpecsWellFormed",
      "status": "True",
      "type": "PrecacheSpecValid"
    }
  ],
  "precaching": {
    "clusters": [
      "cnfdb1" 2
    ],
    "spec": {
      "platformImage": "image.example.io",
      "status": {
        "cnfdb1": "Active"
      }
    }
  }
}

```

1 Displays that the update is in progress.

2 Displays the list of identified clusters.

3. Check the status of the pre-caching job by running the following command on the spoke cluster:

```
$ oc get jobs,pods -n openshift-talm-pre-cache
```

Example output

```

NAME                COMPLETIONS  DURATION  AGE
job.batch/pre-cache  0/1          3m10s    3m10s

```

```

NAME                READY  STATUS  RESTARTS  AGE
pod/pre-cache--1-9bmlr  1/1    Running  0         3m10s

```

4. Check the status of the **ClusterGroupUpgrade** CR by running the following command:

—

```
$ oc get cgu -n ztp-group-du-sno du-upgrade-4918 -o jsonpath='{.status}'
```

Example output

```
"conditions": [
  {
    "lastTransitionTime": "2022-01-27T19:30:41Z",
    "message": "The ClusterGroupUpgrade CR has all clusters compliant with all the
managed policies",
    "reason": "UpgradeCompleted",
    "status": "True",
    "type": "Ready"
  },
  {
    "lastTransitionTime": "2022-01-27T19:28:57Z",
    "message": "Precaching is completed",
    "reason": "PrecachingCompleted",
    "status": "True",
    "type": "PrecachingDone" 1
  }
]
```

1 The pre-cache tasks are done.

18.9. TROUBLESHOOTING THE TOPOLOGY AWARE LIFECYCLE MANAGER

The Topology Aware Lifecycle Manager (TALM) is an OpenShift Container Platform Operator that remediates RHACM policies. When issues occur, use the **oc adm must-gather** command to gather details and logs and to take steps in debugging the issues.

For more information about related topics, see the following documentation:

- [Red Hat Advanced Cluster Management for Kubernetes 2.4 Support Matrix](#)
- [Red Hat Advanced Cluster Management Troubleshooting](#)
- The "Troubleshooting Operator issues" section

18.9.1. General troubleshooting

You can determine the cause of the problem by reviewing the following questions:

- Is the configuration that you are applying supported?
 - Are the RHACM and the OpenShift Container Platform versions compatible?
 - Are the TALM and RHACM versions compatible?
- Which of the following components is causing the problem?
 - [Section 18.9.3, "Managed policies"](#)
 - [Section 18.9.4, "Clusters"](#)

- [Section 18.9.5, “Remediation Strategy”](#)
- [Section 18.9.6, “Topology Aware Lifecycle Manager”](#)

To ensure that the **ClusterGroupUpgrade** configuration is functional, you can do the following:

1. Create the **ClusterGroupUpgrade** CR with the **spec.enable** field set to **false**.
2. Wait for the status to be updated and go through the troubleshooting questions.
3. If everything looks as expected, set the **spec.enable** field to **true** in the **ClusterGroupUpgrade** CR.



WARNING

After you set the **spec.enable** field to **true** in the **ClusterUpgradeGroup** CR, the update procedure starts and you cannot edit the CR's **spec** fields anymore.

18.9.2. Cannot modify the ClusterUpgradeGroup CR

Issue

You cannot edit the **ClusterUpgradeGroup** CR after enabling the update.

Resolution

Restart the procedure by performing the following steps:

1. Remove the old **ClusterGroupUpgrade** CR by running the following command:

```
$ oc delete cgu -n <ClusterGroupUpgradeCR_namespace>
<ClusterGroupUpgradeCR_name>
```

2. Check and fix the existing issues with the managed clusters and policies.
 - a. Ensure that all the clusters are managed clusters and available.
 - b. Ensure that all the policies exist and have the **spec.remediationAction** field set to **inform**.
3. Create a new **ClusterGroupUpgrade** CR with the correct configurations.

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

18.9.3. Managed policies

Checking managed policies on the system

Issue

You want to check if you have the correct managed policies on the system.

Resolution

Run the following command:

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.managedPolicies}'
```

Example output

```
["group-du-sno-validator-du-validator-policy", "policy2-common-nto-sub-policy", "policy3-common-  
ptp-sub-policy"]
```

Checking remediationAction mode

Issue

You want to check if the **remediationAction** field is set to **inform** in the **spec** of the managed policies.

Resolution

Run the following command:

```
$ oc get policies --all-namespaces
```

Example output

NAMESPACE	NAME	REMEDIATION ACTION	COMPLIANCE
STATE	AGE		
default	policy1-common-cluster-version-policy	inform	NonCompliant
5d21h			
default	policy2-common-nto-sub-policy	inform	Compliant 5d21h
default	policy3-common-ptp-sub-policy	inform	NonCompliant 5d21h
default	policy4-common-sriov-sub-policy	inform	NonCompliant 5d21h

Checking policy compliance state

Issue

You want to check the compliance state of policies.

Resolution

Run the following command:

```
$ oc get policies --all-namespaces
```

Example output

NAMESPACE	NAME	REMEDIATION ACTION	COMPLIANCE
STATE	AGE		
default	policy1-common-cluster-version-policy	inform	NonCompliant
5d21h			
default	policy2-common-nto-sub-policy	inform	Compliant 5d21h
default	policy3-common-ptp-sub-policy	inform	NonCompliant 5d21h
default	policy4-common-sriov-sub-policy	inform	NonCompliant 5d21h

18.9.4. Clusters

Checking if managed clusters are present

Issue

You want to check if the clusters in the **ClusterGroupUpgrade** CR are managed clusters.

Resolution

Run the following command:

```
$ oc get managedclusters
```

Example output

NAME	HUB ACCEPTED	MANAGED CLUSTER URLS	JOINED	AVAILABLE
local-cluster	true	https://api.hub.example.com:6443	True	Unknown 13d
spoke1	true	https://api.spoke1.example.com:6443	True	True 13d
spoke3	true	https://api.spoke3.example.com:6443	True	True 27h

1. Alternatively, check the TALM manager logs:

a. Get the name of the TALM manager by running the following command:

```
$ oc get pod -n openshift-operators
```

Example output

NAME	READY	STATUS	RESTARTS	AGE
cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp	2/2	Running	0	45m

b. Check the TALM manager logs by running the following command:

```
$ oc logs -n openshift-operators \
cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp -c manager
```

Example output

```
ERROR controller-runtime.manager.controller.clustergroupupgrade Reconciler error
{"reconciler group": "ran.openshift.io", "reconciler kind": "ClusterGroupUpgrade",
"name": "lab-upgrade", "namespace": "default", "error": "Cluster spoke5555 is not a
ManagedCluster"} 1
sigs.k8s.io/controller-runtime/pkg/internal/controller.
(*Controller).processNextWorkItem
```

1 The error message shows that the cluster is not a managed cluster.

Checking if managed clusters are available

Issue

You want to check if the managed clusters specified in the **ClusterGroupUpgrade** CR are available.

Resolution

Run the following command:

```
$ oc get managedclusters
```

Example output

NAME AGE	HUB ACCEPTED	MANAGED CLUSTER URLS	JOINED	AVAILABLE
local-cluster	true	https://api.hub.testlab.com:6443	True	Unknown 13d
spoke1	true	https://api.spoke1.testlab.com:6443	True	True 13d 1
spoke3	true	https://api.spoke3.testlab.com:6443	True	True 27h 2

1 **2** The value of the **AVAILABLE** field is **True** for the managed clusters.

Checking clusterSelector

Issue

You want to check if the **clusterSelector** field is specified in the **ClusterGroupUpgrade** CR in at least one of the managed clusters.

Resolution

Run the following command:

```
$ oc get managedcluster --selector=upgrade=true 1
```

1 The label for the clusters you want to update is **upgrade:true**.

Example output

NAME AVAILABLE AGE	HUB ACCEPTED	MANAGED CLUSTER URLS	JOINED
spoke1	true	https://api.spoke1.testlab.com:6443	True True 13d
spoke3	true	https://api.spoke3.testlab.com:6443	True True 27h

Checking if canary clusters are present

Issue

You want to check if the canary clusters are present in the list of clusters.

Example ClusterGroupUpgrade CR

```
spec:
  clusters:
  - spoke1
  - spoke3
  clusterSelector:
  - upgrade2=true
```

```
remediationStrategy:
  canaries:
    - spoke3
  maxConcurrency: 2
  timeout: 240
```

Resolution

Run the following commands:

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.clusters}'
```

Example output

```
["spoke1", "spoke3"]
```

1. Check if the canary clusters are present in the list of clusters that match **clusterSelector** labels by running the following command:

```
$ oc get managedcluster --selector=upgrade=true
```

Example output

NAME AGE	HUB ACCEPTED	MANAGED CLUSTER URLS	JOINED	AVAILABLE
spoke1	true	https://api.spoke1.testlab.com:6443	True	True 13d
spoke3	true	https://api.spoke3.testlab.com:6443	True	True 27h



NOTE

A cluster can be present in **spec.clusters** and also be matched by the **spec.clusterSelector** label.

Checking the pre-caching status on spoke clusters

1. Check the status of pre-caching by running the following command on the spoke cluster:

```
$ oc get jobs,pods -n openshift-talo-pre-cache
```

18.9.5. Remediation Strategy

Checking if remediationStrategy is present in the ClusterGroupUpgrade CR

Issue

You want to check if the **remediationStrategy** is present in the **ClusterGroupUpgrade** CR.

Resolution

Run the following command:

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.remediationStrategy}'
```

Example output

```
{"maxConcurrency":2, "timeout":240}
```

Checking if maxConcurrency is specified in the ClusterGroupUpgrade CR

Issue

You want to check if the **maxConcurrency** is specified in the **ClusterGroupUpgrade** CR.

Resolution

Run the following command:

```
$ oc get cgu lab-upgrade -ojsonpath='{.spec.remediationStrategy.maxConcurrency}'
```

Example output

```
2
```

18.9.6. Topology Aware Lifecycle Manager

Checking condition message and status in the ClusterGroupUpgrade CR

Issue

You want to check the value of the **status.conditions** field in the **ClusterGroupUpgrade** CR.

Resolution

Run the following command:

```
$ oc get cgu lab-upgrade -ojsonpath='{.status.conditions}'
```

Example output

```
{"lastTransitionTime":"2022-02-17T22:25:28Z", "message":"The ClusterGroupUpgrade CR has managed policies that are missing:[policyThatDoesntExist]", "reason":"UpgradeCannotStart", "status":"False", "type":"Ready"}
```

Checking corresponding copied policies

Issue

You want to check if every policy from **status.managedPoliciesForUpgrade** has a corresponding policy in **status.copiedPolicies**.

Resolution

Run the following command:

```
$ oc get cgu lab-upgrade -oyaml
```

Example output

```
status:
```

```
...
copiedPolicies:
- lab-upgrade-policy3-common-ntp-sub-policy
managedPoliciesForUpgrade:
- name: policy3-common-ntp-sub-policy
  namespace: default
```

Checking if `status.remediationPlan` was computed

Issue

You want to check if **`status.remediationPlan`** is computed.

Resolution

Run the following command:

```
$ oc get cgu lab-upgrade -ojsonpath='{.status.remediationPlan}'
```

Example output

```
[["spoke2", "spoke3"]]
```

Errors in the TALM manager container

Issue

You want to check the logs of the manager container of TALM.

Resolution

Run the following command:

```
$ oc logs -n openshift-operators \
cluster-group-upgrades-controller-manager-75bcc7484d-8k8xp -c manager
```

Example output

```
ERROR controller-runtime.manager.controller.clustergroupupgrade Reconciler error {"reconciler
group": "ran.openshift.io", "reconciler kind": "ClusterGroupUpgrade", "name": "lab-upgrade",
"namespace": "default", "error": "Cluster spoke5555 is not a ManagedCluster"} 1
sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem
```

1 Displays the error.

Additional resources

- For information about troubleshooting, see [OpenShift Container Platform Troubleshooting Operator Issues](#).
- For more information about using Topology Aware Lifecycle Manager in the ZTP workflow, see [Updating managed policies with Topology Aware Lifecycle Manager](#).

CHAPTER 19. CREATING A PERFORMANCE PROFILE

Learn about the Performance Profile Creator (PPC) and how you can use it to create a performance profile.

19.1. ABOUT THE PERFORMANCE PROFILE CREATOR

The Performance Profile Creator (PPC) is a command-line tool, delivered with the Node Tuning Operator, used to create the performance profile. The tool consumes **must-gather** data from the cluster and several user-supplied profile arguments. The PPC generates a performance profile that is appropriate for your hardware and topology.

The tool is run by one of the following methods:

- Invoking **podman**
- Calling a wrapper script

19.1.1. Gathering data about your cluster using **must-gather**

The Performance Profile Creator (PPC) tool requires **must-gather** data. As a cluster administrator, run **must-gather** to capture information about your cluster.



NOTE

In earlier versions of OpenShift Container Platform, the Performance Addon Operator provided automatic, low latency performance tuning for applications. In OpenShift Container Platform 4.11, these functions are part of the Node Tuning Operator. However, you must still use the **performance-addon-operator-must-gather** image when running the **must-gather** command.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- Access to the Performance Addon Operator **must gather** image.
- The OpenShift CLI (**oc**) installed.

Procedure

1. Navigate to the directory where you want to store the **must-gather** data.
2. Run **must-gather** on your cluster:

```
$ oc adm must-gather --image=<PAO_must_gather_image> --dest-dir=<dir>
```



NOTE

must-gather must be run with the **performance-addon-operator-must-gather** image. The output can optionally be compressed. Compressed output is required if you are running the Performance Profile Creator wrapper script.

Example

```
$ oc adm must-gather --image=registry.redhat.io/openshift4/performance-addon-operator-
must-gather-rhel8:v4.11 --dest-dir=<path_to_must-gather>/must-gather
```

3. Create a compressed file from the **must-gather** directory:

```
$ tar cvaf must-gather.tar.gz must-gather/
```

19.1.2. Running the Performance Profile Creator using podman

As a cluster administrator, you can run **podman** and the Performance Profile Creator to create a performance profile.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- A cluster installed on bare-metal hardware.
- A node with **podman** and OpenShift CLI (**oc**) installed.
- Access to the Node Tuning Operator image.

Procedure

1. Check the machine config pool:

```
$ oc get mcp
```

Example output

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT		
DEGRADEDMACHINECOUNT	AGE			
master	rendered-master-acd1358917e9f98cbdb599aea622d78b	True	False	
False 3	3	3	0	22h
worker-cnf	rendered-worker-cnf-1d871ac76e1951d32b2fe92369879826	False	True	
False 2	1	1	0	22h

2. Use Podman to authenticate to **registry.redhat.io**:

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

```
Username: myrhusername
Password: *****
```

3. Optional: Display help for the PPC tool:

```
$ podman run --rm --entrypoint performance-profile-creator registry.redhat.io/openshift4/ose-
cluster-node-tuning-operator:v4.11 -h
```

Example output

A tool that automates creation of Performance Profiles

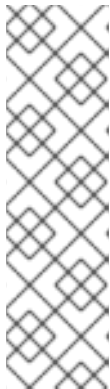
Usage:

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 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
 --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. Run the Performance Profile Creator tool in discovery mode:



NOTE

Discovery mode inspects your cluster using the output from **must-gather**. The output produced includes information on:

- The NUMA cell partitioning with the allocated CPU ids
- Whether hyperthreading is enabled

Using this information you can set appropriate values for some of the arguments supplied to the Performance Profile Creator tool.

```
$ podman run --entrypoint performance-profile-creator -v <path_to_must-gather>/must-gather:/must-gather:z registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.11 --info log --must-gather-dir-path /must-gather
```

**NOTE**

This command uses the performance profile creator as a new entry point to **podman**. It maps the **must-gather** data for the host into the container image and invokes the required user-supplied profile arguments to produce the **my-performance-profile.yaml** file.

The **-v** option can be the path to either:

- The **must-gather** output directory
- An existing directory containing the **must-gather** decompressed tarball

The **info** option requires a value which specifies the output format. Possible values are log and JSON. The JSON format is reserved for debugging.

5. Run **podman**:

```
$ podman run --entrypoint performance-profile-creator -v /must-gather:/must-gather:z
registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:vBranch Build --mcp-
name=worker-cnf --reserved-cpu-count=4 --rt-kernel=true --split-reserved-cpus-across-
numa=false --must-gather-dir-path /must-gather --power-consumption-mode=ultra-low-
latency --offlined-cpu-count=6 > my-performance-profile.yaml
```

**NOTE**

The Performance Profile Creator arguments are shown in the Performance Profile Creator arguments table. The following arguments are required:

- **reserved-cpu-count**
- **mcp-name**
- **rt-kernel**

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**.

6. Review the created YAML file:

```
$ cat my-performance-profile.yaml
```

Example output

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: 2-39,48-79
    offlined: 42-47
    reserved: 0-1,40-41
```



```

machineConfigPoolSelector:
  machineconfiguration.openshift.io/role: worker-cnf
nodeSelector:
  node-role.kubernetes.io/worker-cnf: ""
numa:
  topologyPolicy: restricted
realTimeKernel:
  enabled: true
workloadHints:
  highPowerConsumption: true
  realTime: true

```

7. Apply the generated profile:

```
$ oc apply -f my-performance-profile.yaml
```

19.1.2.1. How to run podman to create a performance profile

The following example illustrates how to run **podman** to create a performance profile with 20 reserved CPUs that are to be split across the NUMA nodes.

Node hardware configuration:

- 80 CPUs
- Hyperthreading enabled
- Two NUMA nodes
- Even numbered CPUs run on NUMA node 0 and odd numbered CPUs run on NUMA node 1

Run **podman** to create the performance profile:

```
$ podman run --entrypoint performance-profile-creator -v /must-gather:/must-gather:z
registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.11 --mcp-name=worker-cnf --
reserved-cpu-count=20 --rt-kernel=true --split-reserved-cpus-across-numa=true --must-gather-dir-
path /must-gather > my-performance-profile.yaml
```

The created profile is described in the following YAML:

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: 10-39,50-79
    reserved: 0-9,40-49
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numa:
    topologyPolicy: restricted
  realTimeKernel:
    enabled: true

```

**NOTE**

In this case, 10 CPUs are reserved on NUMA node 0 and 10 are reserved on NUMA node 1.

19.1.3. Running the Performance Profile Creator wrapper script

The performance profile wrapper script simplifies the running of the Performance Profile Creator (PPC) tool. It hides the complexities associated with running **podman** and specifying the mapping directories and it enables the creation of the performance profile.

Prerequisites

- Access to the Node Tuning Operator image.
- 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-operator:v4.11"
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 -zxvf "${MG_TARBALL}" --directory "${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 "$@"

```

3. Add execute permissions for everyone on this script:

```
$ chmod a+x run-perf-profile-creator.sh
```

4. Optional: Display the **run-perf-profile-creator.sh** command usage:

```
$ ./run-perf-profile-creator.sh -h
```

Expected 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 1
-t          path to a must-gather tarball 2
```

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
--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
```



NOTE

There two types of arguments:

- Wrapper arguments namely **-h**, **-p** and **-t**
- PPC arguments

1 Optional: Specify the Node Tuning Operator image. If not set, the default upstream image is used: **registry.redhat.io/openshift4/ose-cluster-node-tuning-operator:v4.11**.

2 **-t** is a required wrapper script argument and specifies the path to a **must-gather** tarball.

- Run the performance profile creator tool in discovery mode:



NOTE

Discovery mode inspects your cluster using the output from **must-gather**. The output produced includes information on:

- The NUMA cell partitioning with the allocated CPU IDs
- Whether hyperthreading is enabled

Using this information you can set appropriate values for some of the arguments supplied to the Performance Profile Creator tool.

```
$ ./run-perf-profile-creator.sh -t /must-gather/must-gather.tar.gz -- --info=log
```



NOTE

The **info** option requires a value which specifies the output format. Possible values are log and JSON. The JSON format is reserved for debugging.

- Check the machine config pool:

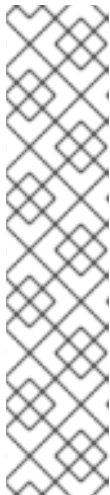
```
$ oc get mcp
```

Example output

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT		
DEGRADEDMACHINECOUNT	AGE			
master	rendered-master-acd1358917e9f98cddb599aea622d78b	True	False	
False	3 3 3 0 22h			
worker-cnf	rendered-worker-cnf-1d871ac76e1951d32b2fe92369879826	False	True	
False	2 1 1 0 22h			

- Create a performance profile:

```
$ ./run-perf-profile-creator.sh -t /must-gather/must-gather.tar.gz -- --mcp-name=worker-cnf --reserved-cpu-count=2 --rt-kernel=true > my-performance-profile.yaml
```



NOTE

The Performance Profile Creator arguments are shown in the Performance Profile Creator arguments table. The following arguments are required:

- **reserved-cpu-count**
- **mcp-name**
- **rt-kernel**

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:

```
$ cat my-performance-profile.yaml
```

Example output

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  cpu:
    isolated: 1-39,41-79
    reserved: 0,40
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numa:
    topologyPolicy: restricted
  realTimeKernel:
    enabled: false
```

9. Apply the generated profile:



NOTE



Install the Node Tuning Operator before applying the profile.


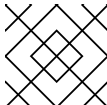
```
$ oc apply -f my-performance-profile.yaml
```

19.1.4. Performance Profile Creator arguments

Table 19.1. Performance Profile Creator arguments

Argument	Description
----------	-------------

Argument	Description
disable-ht	<p>Disable hyperthreading.</p> <p>Possible values: true or false.</p> <p>Default: false.</p> <div>  <p>WARNING</p> <p>If this argument is set to true you should not disable hyperthreading in the BIOS. Disabling hyperthreading is accomplished with a kernel command line argument.</p> </div>
info	<p>This captures cluster information and is used in discovery mode only. Discovery mode also requires the must-gather-dir-path argument. If any other arguments are set they are ignored.</p> <p>Possible values:</p> <ul style="list-style-type: none"> • log • JSON <div>  <p>NOTE</p> <p>These options define the output format with the JSON format being reserved for debugging.</p> </div> <p>Default: log.</p>
mcp-name	<p>MCP name for example worker-cnf corresponding to the target machines. This parameter is required.</p>
must-gather-dir-path	<p>Must gather directory path. This parameter is required.</p> <p>When the user runs the tool with the wrapper script must-gather is supplied by the script itself and the user must not specify it.</p>

Argument	Description
offlined-cpu-count	<p>Number of offlined CPUs.</p> <div>  <div> <p>NOTE</p> <p>This must be a natural number greater than 0. If not enough logical processors are offlined then error messages are logged. The messages are:</p> <div> <p>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]</p> <p>Error: failed to compute the reserved and isolated CPUs: please specify the offlined CPU count in the range [0,1]</p> </div> </div> </div>
power-consumption-mode	<p>The power consumption mode.</p> <p>Possible values:</p> <ul style="list-style-type: none"> • default: CPU partitioning with enabled power management and basic low-latency. • low-latency: Enhanced measures to improve latency figures. • ultra-low-latency: Priority given to optimal latency, at the expense of power management. <p>Default: default.</p>
profile-name	Name of the performance profile to create. Default: performance .
reserved-cpu-count	<p>Number of reserved CPUs. This parameter is required.</p> <div>  <div> <p>NOTE</p> <p>This must be a natural number. A value of 0 is not allowed.</p> </div> </div>
rt-kernel	<p>Enable real-time kernel. This parameter is required.</p> <p>Possible values: true or false.</p>
split-reserved-cpus-across-numa	<p>Split the reserved CPUs across NUMA nodes.</p> <p>Possible values: true or false.</p> <p>Default: false.</p>

Argument	Description
topology-manager-policy	<p>Kubelet Topology Manager policy of the performance profile to be created.</p> <p>Possible values:</p> <ul style="list-style-type: none"> • single-numa-node • best-effort • restricted <p>Default: restricted.</p>
user-level-networking	<p>Run with user level networking (DPDK) enabled.</p> <p>Possible values: true or false.</p> <p>Default: false.</p>

19.2. REFERENCE PERFORMANCE PROFILES

19.2.1. A 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.

A 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: "
realTimeKernel:
  enabled: false
  globallyDisableIrqLoadBalancing: true
```

Insert values that are appropriate for your configuration for the **CPU_ISOLATED**, **CPU_RESERVED**, and **HUGEPAGES_COUNT** keys.

To learn how to create and use performance profiles, see the "Creating a performance profile" page in the "Scalability and performance" section of the OpenShift Container Platform documentation.

19.3. ADDITIONAL RESOURCES

- For more information about the **must-gather** tool, see [Gathering data about your cluster](#).

CHAPTER 20. DEPLOYING DISTRIBUTED UNITS MANUALLY ON SINGLE-NODE OPENSIFT

The procedures in this topic tell you how to manually deploy clusters on a small number of single nodes as a distributed unit (DU) during installation.

The procedures do not describe how to install single-node OpenShift. This can be accomplished through many mechanisms. Rather, they are intended to capture the elements that should be configured as part of the installation process:

- Networking is needed to enable connectivity to the single-node OpenShift DU when the installation is complete.
- Workload partitioning, which can only be configured during installation.
- Additional items that help minimize the potential reboots post installation.

20.1. CONFIGURING THE DISTRIBUTED UNITS (DUS)

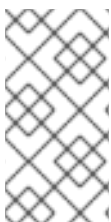
This section describes a set of configurations for an OpenShift Container Platform cluster so that it meets the feature and performance requirements necessary for running a distributed unit (DU) application. Some of this content must be applied during installation and other configurations can be applied post-install.

After you have installed the single-node OpenShift DU, further configuration is needed to enable the platform to carry a DU workload.

The configurations in this section are applied to the cluster after installation in order to configure the cluster for DU workloads.

20.1.1. Enabling workload partitioning

A key feature to enable as part of a single-node OpenShift installation is workload partitioning. This limits the cores allowed to run platform services, maximizing the CPU core for application payloads. You must configure workload partitioning at cluster installation time.



NOTE

You can enable workload partitioning during the cluster installation process only. You cannot disable workload partitioning post-installation. However, you can reconfigure workload partitioning by updating the **cpu** value that you define in the **performanceprofile**, and in the MachineConfig CR in the following procedure.

Procedure

- The base64-encoded content below contains the CPU set that the management workloads are constrained to. This content must be adjusted to match the set specified in the **performanceprofile** and must be accurate for the number of cores on the cluster.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
```

```

name: 02-master-workload-partitioning
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,W2NyaW8ucnVudGltZS53b3JrbG9hZHMubWFuYWdlbWVudF0KYWN0aXZhdGlubl
9hbm5vdGF0aW9uID0gInRhcmdldC53b3JrbG9hZC5vcGVuc2hpZnQuaW8vbWFuYWdlbWVud
dCIKYW5ub3RhdGlubl9wcmVmaXggPSAicmVzb3VyY2VzLndvcmtsb2FkLm9wZW5zaGlmdC5
pbylKW2NyaW8ucnVudGltZS53b3JrbG9hZHMubWFuYWdlbWVudC5yZXNvdXJjZXNdCmNw
dXNoYXJlcjA9IDAKQ1BVcyA9IClwLTESIDUyLTUzlg==
            mode: 420
            overwrite: true
            path: /etc/crio/crio.conf.d/01-workload-partitioning
            user:
              name: root
        - contents:
            source: data:text/plain;charset=utf-
8;base64,ewogICJtYW5hZ2VtZW50IjogewogICAgImNwdXNldCI6IClwLTESNTItNTMiCiAgfQp
9Cg==
            mode: 420
            overwrite: true
            path: /etc/kubernetes/openshift-workload-pinning
            user:
              name: root

```

- The contents of **/etc/crio/crio.conf.d/01-workload-partitioning** should look like this:

```

[crio.runtime.workloads.management]
activation_annotation = "target.workload.openshift.io/management"
annotation_prefix = "resources.workload.openshift.io"
[crio.runtime.workloads.management.resources]
cpushares = 0
CPUs = "0-1, 52-53" ❶

```

- ❶ The **CPUs** value varies based on the installation.

If Hyper-Threading is enabled, specify both threads of each core. The **CPUs** value must match the reserved CPU set specified in the performance profile.

This content should be base64 encoded and provided in the **01-workload-partitioning-content** in the manifest above.

- The contents of **/etc/kubernetes/openshift-workload-pinning** should look like this:

```

{
  "management": {
    "cpuset": "0-1,52-53" ❶
  }
}

```

233

container-specific mounts

```

[Service]
Type=oneshot
RemainAfterExit=yes
RuntimeDirectory=container-mount-namespace
Environment=RUNTIME_DIRECTORY=%t/container-mount-namespace
Environment=BIND_POINT=%t/container-mount-namespace/mnt
ExecStartPre=bash -c "findmnt ${RUNTIME_DIRECTORY} || mount --make-unbindable --bind
${RUNTIME_DIRECTORY} ${RUNTIME_DIRECTORY}"
ExecStartPre=touch ${BIND_POINT}
ExecStart=unshare --mount=${BIND_POINT} --propagation slave mount --make-rshared /
ExecStop=umount -R ${RUNTIME_DIRECTORY}
enabled: true
name: container-mount-namespace.service
- dropins:
- contents: |
    [Unit]
    Wants=container-mount-namespace.service
    After=container-mount-namespace.service

[Service]
ExecStartPre=/usr/local/bin/extractExecStart %n /%t/%N-execstart.env ORIG_EXECSTART
EnvironmentFile=-/%t/%N-execstart.env
ExecStart=
ExecStart=bash -c "nsenter --mount=%t/container-mount-namespace/mnt \
    ${ORIG_EXECSTART}"
name: 90-container-mount-namespace.conf
name: crio.service
- dropins:
- contents: |
    [Unit]
    Wants=container-mount-namespace.service
    After=container-mount-namespace.service

[Service]
ExecStartPre=/usr/local/bin/extractExecStart %n /%t/%N-execstart.env ORIG_EXECSTART
EnvironmentFile=-/%t/%N-execstart.env
ExecStart=
ExecStart=bash -c "nsenter --mount=%t/container-mount-namespace/mnt \
    ${ORIG_EXECSTART} --housekeeping-interval=30s"
name: 90-container-mount-namespace.conf
- contents: |
    [Service]
    Environment="OPENSIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION=60s"
    Environment="OPENSIFT_EVICTION_MONITORING_PERIOD_DURATION=30s"
name: 30-kubelet-interval-tuning.conf
name: kubelet.service

```

20.1.3. Enabling Stream Control Transmission Protocol (SCTP)

SCTP is a key protocol used in RAN applications. This **MachineConfig** object adds the SCTP kernel module to the node to enable this protocol.

Procedure

- No configuration changes are needed. Use the provided settings:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: load-sctp-module
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
        - contents:
            source: data:,
            verification: {}
          filesystem: root
          mode: 420
          path: /etc/modprobe.d/sctp-blacklist.conf
        - contents:
            source: data:text/plain;charset=utf-8,sctp
          filesystem: root
          mode: 420
          path: /etc/modules-load.d/sctp-load.conf
```

20.1.4. Creating OperatorGroups for Operators

This configuration is provided to enable addition of the Operators needed to configure the platform post-installation. It adds the **Namespace** and **OperatorGroup** objects for the Local Storage Operator, Logging Operator, PTP Operator, and SR-IOV Network Operator.

Procedure

- No configuration changes are needed. Use the provided settings:

Local Storage Operator

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    workload.openshift.io/allowed: management
  name: openshift-local-storage
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-local-storage
  namespace: openshift-local-storage
spec:
  targetNamespaces:
    - openshift-local-storage
```

Logging Operator

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    workload.openshift.io/allowed: management
  name: openshift-logging
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: cluster-logging
  namespace: openshift-logging
spec:
  targetNamespaces:
    - openshift-logging
```

PTP Operator

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    workload.openshift.io/allowed: management
  labels:
    openshift.io/cluster-monitoring: "true"
  name: openshift-ptp
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ptp-operators
  namespace: openshift-ptp
spec:
  targetNamespaces:
    - openshift-ptp
```

SR-IOV Network Operator

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    workload.openshift.io/allowed: management
  name: openshift-sriov-network-operator
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
```



```
spec:
  targetNamespaces:
    - openshift-sriov-network-operator
```

20.1.5. Subscribing to the Operators

The subscription provides the location to download the Operators needed for platform configuration.

Procedure

- Use the following example to configure the subscription:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: cluster-logging
  namespace: openshift-logging
spec:
  channel: "stable" 1
  name: cluster-logging
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual 2
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: local-storage-operator
  namespace: openshift-local-storage
spec:
  channel: "stable" 3
  installPlanApproval: Automatic
  name: local-storage-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ptp-operator-subscription
  namespace: openshift-ptp
spec:
  channel: "stable" 4
  name: ptp-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Manual
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: sriov-network-operator-subscription
  namespace: openshift-sriov-network-operator
spec:
```

```
channel: "stable" 5
name: sriov-network-operator
source: redhat-operators
sourceNamespace: openshift-marketplace
installPlanApproval: Manual
```

- 1 Specify the channel to get the **cluster-logging** Operator.
- 2 Specify **Manual** or **Automatic**. In **Automatic** mode, the Operator automatically updates to the latest versions in the channel as they become available in the registry. In **Manual** mode, new Operator versions are installed only after they are explicitly approved.
- 3 Specify the channel to get the **local-storage-operator** Operator.
- 4 Specify the channel to get the **ptp-operator** Operator.
- 5 Specify the channel to get the **sriov-network-operator** Operator.

20.1.6. Configuring logging locally and forwarding

To be able to debug a single node distributed unit (DU), logs need to be stored for further analysis.

Procedure

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

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging 1
metadata:
  name: instance
  namespace: openshift-logging
spec:
  collection:
    logs:
      fluentd: {}
      type: fluentd
  curation:
    type: "curator"
    curator:
      schedule: "30 3 * * *"
  managementState: Managed
---
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder 2
metadata:
  name: instance
  namespace: openshift-logging
spec:
  inputs:
    - infrastructure: {}
  outputs:
    - name: kafka-open
      type: kafka
      url: tcp://10.46.55.190:9092/test 3
```

```

pipelines:
- inputRefs:
  - audit
  name: audit-logs
  outputRefs:
  - kafka-open
- inputRefs:
  - infrastructure
  name: infrastructure-logs
  outputRefs:
  - kafka-open

```

- 1 Updates the existing instance or creates the instance if it does not exist.
- 2 Updates the existing instance or creates the instance if it does not exist.
- 3 Specifies the destination of the kafka server.

20.1.7. Configuring the Node Tuning Operator

This is a key configuration for the single node distributed unit (DU). Many of the real-time capabilities and service assurance are configured here.

Procedure

- Configure the performance profile using the following example:

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: perfprofile-policy
spec:
  additionalKernelArgs:
    - idle=poll
    - rcupdate.rcu_normal_after_boot=0
  cpu:
    isolated: 2-19,22-39 1
    reserved: 0-1,20-21 2
  hugepages:
    defaultHugepagesSize: 1G
  pages:
    - count: 32 3
    size: 1G 4
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/master: ""
  net:
    userLevelNetworking: true 5
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numa:
    topologyPolicy: restricted
  realTimeKernel:
    enabled: true 6

```

- 1 Set the isolated CPUs. Ensure all of the HT pairs match.
- 2 Set the reserved CPUs. In this case, a hyperthreaded pair is allocated on NUMA 0 and a pair on NUMA 1.
- 3 Set the huge page size.
- 4 Set the huge page number.
- 5 Set to **true** to isolate the CPUs from networking interrupts.
- 6 Set to **true** to install the real-time Linux kernel.

20.1.8. Configuring Precision Time Protocol (PTP)

In the far edge, the RAN uses PTP to synchronize the systems.

Procedure

- Configure PTP using the following example:

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: du-ptp-slave
  namespace: openshift-ptp
spec:
  profile:
    - interface: ens5f0 1
      name: slave
      phc2sysOpts: -a -r -n 24
      ptp4lConf: |
        [global]
        #
        # Default Data Set
        #
        twoStepFlag 1
        slaveOnly 0
        priority1 128
        priority2 128
        domainNumber 24
        #utc_offset 37
        clockClass 248
        clockAccuracy 0xFE
        offsetScaledLogVariance 0xFFFF
        free_running 0
        freq_est_interval 1
        dscp_event 0
        dscp_general 0
        dataset_comparison ieee1588
        G.8275.defaultDS.localPriority 128
        #
        # Port Data Set
        #
        logAnnounceInterval -3
```

```

logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval -4
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval 4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 0.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l

```

```

#
# Default interface options
#
clock_type OC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 0
#
# Clock description
#
productDescription ;;
revisionData ;;
manufacturerIdentity 00:00:00
userDescription ;
timeSource 0xA0
ptp4IOpts: -2 -s --summary_interval -4
recommend:
- match:
  - nodeLabel: node-role.kubernetes.io/master
priority: 4
profile: slave

```

- 1 Sets the interface used for PTP.

20.1.9. Disabling Network Time Protocol (NTP)

After the system is configured for Precision Time Protocol (PTP), you need to remove NTP to prevent it from impacting the system clock.

Procedure

- No configuration changes are needed. Use the provided settings:

```

apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: disable-chronyd
spec:
  config:
    systemd:
      units:
        - contents: |
            [Unit]
            Description=NTP client/server
            Documentation=man:chronyd(8) man:chrony.conf(5)
            After=ntpdate.service sntp.service ntpd.service

```

```

Conflicts=ntpd.service systemd-timesyncd.service
ConditionCapability=CAP_SYS_TIME
[Service]
Type=forking
PIDFile=/run/chrony/chronyd.pid
EnvironmentFile=-/etc/sysconfig/chronyd
ExecStart=/usr/sbin/chronyd $OPTIONS
ExecStartPost=/usr/libexec/chrony-helper update-daemon
PrivateTmp=yes
ProtectHome=yes
ProtectSystem=full
[Install]
WantedBy=multi-user.target
enabled: false
name: chronyd.service
ignition:
version: 2.2.0

```

20.1.10. Configuring single root I/O virtualization (SR-IOV)

SR-IOV is commonly used to enable the fronthaul and the midhaul networks.

Procedure

- Use the following configuration to configure SRIOV on a single node distributed unit (DU). Note that the first custom resource (CR) is required. The following CRs are examples.

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  configDaemonNodeSelector:
    node-role.kubernetes.io/master: ""
  disableDrain: true
  enableInjector: true
  enableOperatorWebhook: true
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-nw-du-mh
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: openshift-sriov-network-operator
  resourceName: du_mh
  vlan: 150 1
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-nnp-du-mh
  namespace: openshift-sriov-network-operator
spec:

```

```

deviceType: vfio-pci ❷
isRdma: false
nicSelector:
  pfNames:
    - ens7f0 ❸
nodeSelector:
  node-role.kubernetes.io/master: ""
numVfs: 8 ❹
priority: 10
resourceName: du_mh
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-nw-du-fh
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: openshift-sriov-network-operator
  resourceName: du_fh
  vlan: 140 ❺
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-nnp-du-fh
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice ❻
  isRdma: true
  nicSelector:
    pfNames:
      - ens5f0 ❼
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numVfs: 8 ❽
  priority: 10
  resourceName: du_fh

```

- ❶ Specifies the VLAN for the midhaul network.
- ❷ Select either **vfio-pci** or **netdevice**, as needed.
- ❸ Specifies the interface connected to the midhaul network.
- ❹ Specifies the number of VFs for the midhaul network.
- ❺ The VLAN for the fronthaul network.
- ❻ Select either **vfio-pci** or **netdevice**, as needed.
- ❼ Specifies the interface connected to the fronthaul network.
- ❽ Specifies the number of VFs for the fronthaul network.

20.1.11. Disabling the console Operator

The console-operator installs and maintains the web console on a cluster. When the node is centrally managed the Operator is not needed and makes space for application workloads.

Procedure

- You can disable the Operator using the following configuration file. No configuration changes are needed. Use the provided settings:

```
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "false"
    include.release.openshift.io/self-managed-high-availability: "false"
    include.release.openshift.io/single-node-developer: "false"
    release.openshift.io/create-only: "true"
  name: cluster
spec:
  logLevel: Normal
  managementState: Removed
  operatorLogLevel: Normal
```

20.2. APPLYING THE DISTRIBUTED UNIT (DU) CONFIGURATION TO A SINGLE-NODE OPENSIFT CLUSTER

Perform the following tasks to configure a single-node cluster for a DU:

- Apply the required extra installation manifests at installation time.
- Apply the post-install configuration custom resources (CRs).

20.2.1. Applying the extra installation manifests

To apply the distributed unit (DU) configuration to the single-node cluster, the following extra installation manifests need to be included during installation:

- Enable workload partitioning.
- Other **MachineConfig** objects – There is a set of **MachineConfig** custom resources (CRs) included by default. You can choose to include these additional **MachineConfig** CRs that are unique to their environment. It is recommended, but not required, to apply these CRs during installation in order to minimize the number of reboots that can occur during post-install configuration.

20.2.2. Applying the post-install configuration custom resources (CRs)

- After OpenShift Container Platform is installed on the cluster, use the following command to apply the CRs you configured for the distributed units (DUs):

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

CHAPTER 21. VALIDATING CLUSTER TUNING FOR VDU APPLICATION WORKLOADS

Before you can deploy virtualized distributed unit (vDU) applications, you need to tune and configure the cluster host BIOS and various other cluster configuration settings. Use the following information to validate the cluster configuration to support vDU workloads.

Additional resources

- For a complete reference to configuring single-node OpenShift clusters tuned for vDU application deployments, see [Deploying distributed units manually on single-node OpenShift](#).

21.1. RECOMMENDED BIOS CONFIGURATION FOR VDU CLUSTER HOSTS

Use the following table as the basis to configure the cluster host BIOS for vDU applications running on OpenShift Container Platform 4.11.



NOTE

The following table is a general recommendation for vDU cluster host BIOS configuration. Exact BIOS settings will depend on your requirements and specific hardware platform. Automatic setting of BIOS is not handled by the zero touch provisioning pipeline.

Table 21.1. Recommended cluster host BIOS settings

BIOS setting	Configuration	Description
HyperTransport (HT)	Enabled	HyperTransport (HT) bus is a bus technology developed by AMD. HT provides a high-speed link between the components in the host memory and other system peripherals.
UEFI	Enabled	Enable booting from UEFI for the vDU host.
CPU Power and Performance Policy	Performance	Set CPU Power and Performance Policy to optimize the system for performance over energy efficiency.
Uncore Frequency Scaling	Disabled	Disable Uncore Frequency Scaling to prevent the voltage and frequency of non-core parts of the CPU from being set independently.
Uncore Frequency	Maximum	Sets the non-core parts of the CPU such as cache and memory controller to their maximum possible frequency of operation.
Performance P-limit	Disabled	Disable Performance P-limit to prevent the Uncore frequency coordination of processors.

BIOS setting	Configuration	Description
Enhanced Intel® SpeedStep Tech	Enabled	Enable Enhanced Intel SpeedStep to allow the system to dynamically adjust processor voltage and core frequency that decreases power consumption and heat production in the host.
Intel® Turbo Boost Technology	Enabled	Enable Turbo Boost Technology for Intel-based CPUs to automatically allow processor cores to run faster than the rated operating frequency if they are operating below power, current, and temperature specification limits.
Intel Configurable TDP	Enabled	Enables Thermal Design Power (TDP) for the CPU.
Configurable TDP Level	Level 2	TDP level sets the CPU power consumption required for a particular performance rating. TDP level 2 sets the CPU to the most stable performance level at the cost of power consumption.
Energy Efficient Turbo	Disabled	Disable Energy Efficient Turbo to prevent the processor from using an energy-efficiency based policy.
Hardware P-States	Disabled	Disable P-states (performance states) to optimize the operating system and CPU for performance over power consumption.
Package C-State	C0/C1 state	Use C0 or C1 states to set the processor to a fully active state (C0) or to stop CPU internal clocks running in software (C1).
C1E	Disabled	CPU Enhanced Halt (C1E) is a power saving feature in Intel chips. Disabling C1E prevents the operating system from sending a halt command to the CPU when inactive.
Processor C6	Disabled	C6 power-saving is a CPU feature that automatically disables idle CPU cores and cache. Disabling C6 improves system performance.
Sub-NUMA Clustering	Disabled	Sub-NUMA clustering divides the processor cores, cache, and memory into multiple NUMA domains. Disabling this option can increase performance for latency-sensitive workloads.

**NOTE**

Enable global SR-IOV and VT-d settings in the BIOS for the host. These settings are relevant to bare-metal environments.

21.2. RECOMMENDED CLUSTER CONFIGURATIONS TO RUN VDU APPLICATIONS

Clusters running virtualized distributed unit (vDU) applications require a highly tuned and optimized configuration. The following information describes the various elements that you require to support vDU workloads in OpenShift Container Platform 4.11 clusters.

21.2.1. Recommended cluster MachineConfig CRs

The following **MachineConfig** CRs configure the cluster host:

Table 21.2. Recommended MachineConfig CRs

CR filename	Description
02-workload-partitioning.yaml	Configures workload partitioning for the cluster. Apply this MachineConfig CR when you install the cluster.
MachineConfigSctp.yaml	Loads the SCTP kernel module. This MachineConfig CR is optional and can be omitted if you do not require this kernel module.
MachineConfigContainerMountNS.yaml	Configures the container mount namespace and kubelet conf.
MachineConfigAcceleratedStartup.yaml	Configures accelerated startup for the cluster.
06-kdump-master.yaml, 06-kdump-worker.yaml	Configures kdump for the cluster.

21.2.2. Recommended cluster Operators

The following Operators are required for clusters running vDU applications and are a part of the baseline reference configuration:

- Node Tuning Operator (NTO). NTO packages functionality that was previously delivered with the Performance Addon Operator, which is now a part of NTO.
- PTP Operator
- SR-IOV Network Operator
- Red Hat OpenShift Logging Operator
- Local Storage Operator

21.2.3. Recommended cluster kernel configuration

Always use the latest supported realtime kernel version in your cluster. You should also ensure that the following configurations are applied in the cluster:

1. Ensure the following **additionalKernelArgs** are set in the cluster performance profile:

```
spec:
  additionalKernelArgs:
```

```
- "idle=poll"
- "rcupdate.rcu_normal_after_boot=0"
- "efi=runtime"
```

2. Ensure that the **performance-patch** profile in the **Tuned** CR configures the correct CPU isolation set that matches the **isolated** CPU set in the related **PerformanceProfile** CR, for example:

```
spec:
  profile:
    - name: performance-patch
      # The 'include' line must match the associated PerformanceProfile name
      # And the cmdline_crash CPU set must match the 'isolated' set in the associated
      PerformanceProfile
    data: |
      [main]
      summary=Configuration changes profile inherited from performance created tuned
      include=openshift-node-performance-openshift-node-performance-profile
      [bootloader]
      cmdline_crash=nohz_full=2-51,54-103 1
      [sysctl]
      kernel.timer_migration=1
      [scheduler]
      group.ice-ptp=0:f:10*:ice-ptp.*
      [service]
      service.stalld=start,enable
      service.chrond=stop,disable
```

- 1** Listed CPUs depend on the host hardware configuration, specifically the number of available CPUs in the system and the CPU topology.

21.2.4. Checking the realtime kernel version

Always use the latest version of the realtime kernel in your OpenShift Container Platform clusters. If you are unsure about the kernel version that is in use in the cluster, you can compare the current realtime kernel version to the release version with the following procedure.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You are logged in as a user with **cluster-admin** privileges.
- You have installed **podman**.

Procedure

1. Run the following command to get the cluster version:

```
$ OCP_VERSION=$(oc get clusterversion version -o jsonpath='{.status.desired.version}'
{"\n"})
```

2. Get the release image SHA number:

■

```
$ DTK_IMAGE=$(oc adm release info --image-for=driver-toolkit quay.io/openshift-release-dev/ocp-release:$OCP_VERSION-x86_64)
```

3. Run the release image container and extract the kernel version that is packaged with cluster's current release:

```
$ podman run --rm $DTK_IMAGE rpm -qa | grep 'kernel-rt-core-' | sed 's#kernel-rt-core-##'
```

Example output

```
4.18.0-305.49.1.rt7.121.el8_4.x86_64
```

This is the default realtime kernel version that ships with the release.



NOTE

The realtime kernel is denoted by the string **.rt** in the kernel version.

Verification

Check that the kernel version listed for the cluster's current release matches actual realtime kernel that is running in the cluster. Run the following commands to check the running realtime kernel version:

1. Open a remote shell connection to the cluster node:

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

2. Check the realtime kernel version:

```
sh-4.4# uname -r
```

Example output

```
4.18.0-305.49.1.rt7.121.el8_4.x86_64
```

21.3. CHECKING THAT THE RECOMMENDED CLUSTER CONFIGURATIONS ARE APPLIED

You can check that clusters are running the correct configuration. The following procedure describes how to check the various configurations that you require to deploy a DU application in OpenShift Container Platform 4.11 clusters.

Prerequisites

- You have deployed a cluster and tuned it for vDU workloads.
- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.

Procedure

1. Check that the default Operator Hub sources are disabled. Run the following command:

```
$ oc get operatorhub cluster -o yaml
```

Example output

```
spec:
  disableAllDefaultSources: true
```

2. Check that all required **CatalogSource** resources are annotated for workload partitioning (**PreferredDuringScheduling**) by running the following command:

```
$ oc get catalogsource -A -o jsonpath='{range .items[*]}{.metadata.name}" -- "{.metadata.annotations.target\workload\openshift\io/management}'{"\n"}{end}'
```

Example output

```
certified-operators -- {"effect": "PreferredDuringScheduling"}
community-operators -- {"effect": "PreferredDuringScheduling"}
ran-operators ❶
redhat-marketplace -- {"effect": "PreferredDuringScheduling"}
redhat-operators -- {"effect": "PreferredDuringScheduling"}
```

- ❶ **CatalogSource** resources that are not annotated are also returned. In this example, the **ran-operators CatalogSource** resource is not annotated and does not have the **PreferredDuringScheduling** annotation.



NOTE

In a properly configured vDU cluster, only a single annotated catalog source is listed.

3. Check that all applicable OpenShift Container Platform Operator namespaces are annotated for workload partitioning. This includes all Operators installed with core OpenShift Container Platform and the set of additional Operators included in the reference DU tuning configuration. Run the following command:

```
$ oc get namespaces -A -o jsonpath='{range .items[*]}{.metadata.name}" -- "{.metadata.annotations.workload\openshift\io/allowed}'{"\n"}{end}'
```

Example output

```
default --
openshift-apiserver -- management
openshift-apiserver-operator -- management
openshift-authentication -- management
openshift-authentication-operator -- management
```

**IMPORTANT**

Additional Operators must not be annotated for workload partitioning. In the output from the previous command, additional Operators should be listed without any value on the right-hand side of the `--` separator.

4. Check that the **ClusterLogging** configuration is correct. Run the following commands:

- a. Validate that the appropriate input and output logs are configured:

```
$ oc get -n openshift-logging ClusterLogForwarder instance -o yaml
```

Example output

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  creationTimestamp: "2022-07-19T21:51:41Z"
  generation: 1
  name: instance
  namespace: openshift-logging
  resourceVersion: "1030342"
  uid: 8c1a842d-80c5-447a-9150-40350bdf40f0
spec:
  inputs:
    - infrastructure: {}
      name: infra-logs
  outputs:
    - name: kafka-open
      type: kafka
      url: tcp://10.46.55.190:9092/test
  pipelines:
    - inputRefs:
        - audit
      name: audit-logs
      outputRefs:
        - kafka-open
    - inputRefs:
        - infrastructure
      name: infrastructure-logs
      outputRefs:
        - kafka-open
  ...
```

- b. Check that the curation schedule is appropriate for your application:

```
$ oc get -n openshift-logging clusterloggings.logging.openshift.io instance -o yaml
```

Example output

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  creationTimestamp: "2022-07-07T18:22:56Z"
```



```

generation: 1
name: instance
namespace: openshift-logging
resourceVersion: "235796"
uid: ef67b9b8-0e65-4a10-88ff-ec06922ea796
spec:
  collection:
    logs:
      fluentd: {}
      type: fluentd
    curation:
      curator:
        schedule: 30 3 * * *
        type: curator
      managementState: Managed
  ...

```

5. Check that the web console is disabled (**managementState: Removed**) by running the following command:

```
$ oc get consoles.operator.openshift.io cluster -o jsonpath="{ .spec.managementState }"
```

Example output

```
Removed
```

6. Check that **chronyd** is disabled on the cluster node by running the following commands:

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

Check the status of **chronyd** on the node:

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

```
sh-4.4# systemctl status chronyd
```

Example output

```

• chronyd.service - NTP client/server
  Loaded: loaded (/usr/lib/systemd/system/chronyd.service; disabled; vendor preset:
enabled)
  Active: inactive (dead)
  Docs: man:chronyd(8)
        man:chrony.conf(5)

```

7. Check that the PTP interface is successfully synchronized to the primary clock using a remote shell connection to the **linuxptp-daemon** container and the PTP Management Client (**pmc**) tool:
 - a. Set the **\$PTP_POD_NAME** variable with the name of the **linuxptp-daemon** pod by running the following command:

```
$ PTP_POD_NAME=$(oc get pods -n openshift-ptp -l app=linuxptp-daemon -o name)
```

- b. Run the following command to check the sync status of the PTP device:

```
$ oc -n openshift-ptp rsh -c linuxptp-daemon-container ${PTP_POD_NAME} pmc -u -f
/var/run/ptp4l.0.config -b 0 'GET PORT_DATA_SET'
```

Example output

```
sending: GET PORT_DATA_SET
3cecef.ffe.7a7020-1 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
portIdentity      3cecef.ffe.7a7020-1
portState         SLAVE
logMinDelayReqInterval -4
peerMeanPathDelay 0
logAnnounceInterval 1
announceReceiptTimeout 3
logSyncInterval   0
delayMechanism     1
logMinPdelayReqInterval 0
versionNumber      2
3cecef.ffe.7a7020-2 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
portIdentity      3cecef.ffe.7a7020-2
portState         LISTENING
logMinDelayReqInterval 0
peerMeanPathDelay 0
logAnnounceInterval 1
announceReceiptTimeout 3
logSyncInterval   0
delayMechanism     1
logMinPdelayReqInterval 0
versionNumber      2
```

- c. Run the following **pmc** command to check the PTP clock status:

```
$ oc -n openshift-ptp rsh -c linuxptp-daemon-container ${PTP_POD_NAME} pmc -u -f
/var/run/ptp4l.0.config -b 0 'GET TIME_STATUS_NP'
```

Example output

```
sending: GET TIME_STATUS_NP
3cecef.ffe.7a7020-0 seq 0 RESPONSE MANAGEMENT TIME_STATUS_NP
master_offset      10 1
ingress_time       1657275432697400530
cumulativeScaledRateOffset +0.000000000
scaledLastGmPhaseChange 0
gmTimeBaseIndicator 0
lastGmPhaseChange 0x0000'0000000000000000.0000
gmPresent          true 2
gmIdentity          3c2c30.fff.670e00
```

1 **master_offset** should be between -100 and 100 ns.

2 Indicates that the PTP clock is synchronized to a master, and the local clock is not the grandmaster clock.

- d. Check that the expected **master offset** value corresponding to the value in `/var/run/ptp4l.0.config` is found in the **linuxptp-daemon-container** log:

```
$ oc logs $PTP_POD_NAME -n openshift-ptp -c linuxptp-daemon-container
```

Example output

```
phc2sys[56020.341]: [ptp4l.1.config] CLOCK_REALTIME phc offset -1731092 s2 freq -
1546242 delay 497
ptp4l[56020.390]: [ptp4l.1.config] master offset -2 s2 freq -5863 path delay 541
ptp4l[56020.390]: [ptp4l.0.config] master offset -8 s2 freq -10699 path delay 533
```

8. Check that the SR-IOV configuration is correct by running the following commands:

- a. Check that the **disableDrain** value in the **SriovOperatorConfig** resource is set to **true**:

```
$ oc get sriovoperatorconfig -n openshift-sriov-network-operator default -o jsonpath="
{.spec.disableDrain}"
```

Example output

```
true
```

- b. Check that the **SriovNetworkNodeState** sync status is **Succeeded** by running the following command:

```
$ oc get SriovNetworkNodeStates -n openshift-sriov-network-operator -o jsonpath="
{.items[*].status.syncStatus}"
```

Example output

```
Succeeded
```

- c. Verify that the expected number and configuration of virtual functions (**Vfs**) under each interface configured for SR-IOV is present and correct in the **.status.interfaces** field. For example:

```
$ oc get SriovNetworkNodeStates -n openshift-sriov-network-operator -o yaml
```

Example output

```
apiVersion: v1
items:
- apiVersion: sriovnetwork.openshift.io/v1
  kind: SriovNetworkNodeState
  ...
  status:
    interfaces:
      ...
      - Vfs:
        - deviceID: 154c
          driver: vfio-pci
```

```

pciAddress: 0000:3b:0a.0
vendor: "8086"
vfID: 0
- deviceID: 154c
  driver: vfio-pci
  pciAddress: 0000:3b:0a.1
  vendor: "8086"
  vfID: 1
- deviceID: 154c
  driver: vfio-pci
  pciAddress: 0000:3b:0a.2
  vendor: "8086"
  vfID: 2
- deviceID: 154c
  driver: vfio-pci
  pciAddress: 0000:3b:0a.3
  vendor: "8086"
  vfID: 3
- deviceID: 154c
  driver: vfio-pci
  pciAddress: 0000:3b:0a.4
  vendor: "8086"
  vfID: 4
- deviceID: 154c
  driver: vfio-pci
  pciAddress: 0000:3b:0a.5
  vendor: "8086"
  vfID: 5
- deviceID: 154c
  driver: vfio-pci
  pciAddress: 0000:3b:0a.6
  vendor: "8086"
  vfID: 6
- deviceID: 154c
  driver: vfio-pci
  pciAddress: 0000:3b:0a.7
  vendor: "8086"
  vfID: 7

```

9. Check that the cluster performance profile is correct. The **cpu** and **hugepages** sections will vary depending on your hardware configuration. Run the following command:

```
$ oc get PerformanceProfile openshift-node-performance-profile -o yaml
```

Example output

```

apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  creationTimestamp: "2022-07-19T21:51:31Z"
  finalizers:
    - foreground-deletion
  generation: 1
  name: openshift-node-performance-profile
  resourceVersion: "33558"

```

```

uid: 217958c0-9122-4c62-9d4d-fdc27c31118c
spec:
  additionalKernelArgs:
    - idle=poll
    - rcupdate.rcu_normal_after_boot=0
    - efi=runtime
  cpu:
    isolated: 2-51,54-103
    reserved: 0-1,52-53
  hugepages:
    defaultHugepagesSize: 1G
  pages:
    - count: 32
    size: 1G
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/master: ""
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numa:
    topologyPolicy: restricted
  realTimeKernel:
    enabled: true
  status:
    conditions:
      - lastHeartbeatTime: "2022-07-19T21:51:31Z"
        lastTransitionTime: "2022-07-19T21:51:31Z"
        status: "True"
        type: Available
      - lastHeartbeatTime: "2022-07-19T21:51:31Z"
        lastTransitionTime: "2022-07-19T21:51:31Z"
        status: "True"
        type: Upgradeable
      - lastHeartbeatTime: "2022-07-19T21:51:31Z"
        lastTransitionTime: "2022-07-19T21:51:31Z"
        status: "False"
        type: Progressing
      - lastHeartbeatTime: "2022-07-19T21:51:31Z"
        lastTransitionTime: "2022-07-19T21:51:31Z"
        status: "False"
        type: Degraded
    runtimeClass: performance-openshift-node-performance-profile
    tuned: openshift-cluster-node-tuning-operator/openshift-node-performance-openshift-node-
    performance-profile

```



NOTE

CPU settings are dependent on the number of cores available on the server and should align with workload partitioning settings. **hugepages** configuration is server and application dependent.

10. Check that the **PerformanceProfile** was successfully applied to the cluster by running the following command:

```
$ oc get performanceprofile openshift-node-performance-profile -o jsonpath="{range
.status.conditions[*]}{ @.type }{' -- '}{@.status}{'\n'}{end}"
```

Example output

```
Available -- True
Upgradeable -- True
Progressing -- False
Degraded -- False
```

11. Check the **Tuned** performance patch settings by running the following command:

```
$ oc get tuned.tuned.openshift.io -n openshift-cluster-node-tuning-operator performance-
patch -o yaml
```

Example output

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  creationTimestamp: "2022-07-18T10:33:52Z"
  generation: 1
  name: performance-patch
  namespace: openshift-cluster-node-tuning-operator
  resourceVersion: "34024"
  uid: f9799811-f744-4179-bf00-32d4436c08fd
spec:
  profile:
    - data: |
        [main]
        summary=Configuration changes profile inherited from performance created tuned
        include=openshift-node-performance-openshift-node-performance-profile
        [bootloader]
        cmdline_crash=nohz_full=2-23,26-47 1
        [sysctl]
        kernel.timer_migration=1
        [scheduler]
        group.ice-ptp=0:f:10:*:ice-ptp.*
        [service]
        service.stalld=start,enable
        service.chrond=stop,disable
        name: performance-patch
      recommend:
        - machineConfigLabels:
            machineconfiguration.openshift.io/role: master
          priority: 19
          profile: performance-patch
```

- 1 The cpu list in **cmdline=nohz_full=** will vary based on your hardware configuration.

12. Check that cluster networking diagnostics are disabled by running the following command:

```
$ oc get networks.operator.openshift.io cluster -o
jsonpath='{.spec.disableNetworkDiagnostics}'
```

Example output

```
true
```

13. Check that the **Kubelet** housekeeping interval is tuned to slower rate. This is set in the **containerMountNS** machine config. Run the following command:

```
$ oc describe machineconfig container-mount-namespace-and-kubelet-conf-master | grep
OPENSIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION
```

Example output

```
Environment="OPENSIFT_MAX_HOUSEKEEPING_INTERVAL_DURATION=60s"
```

14. Check that Grafana and **alertManagerMain** are disabled and that the Prometheus retention period is set to 24h by running the following command:

```
$ oc get configmap cluster-monitoring-config -n openshift-monitoring -o jsonpath="{
.data.config\.yaml }"
```

Example output

```
grafana:
  enabled: false
alertmanagerMain:
  enabled: false
prometheusK8s:
  retention: 24h
```

- a. Use the following commands to verify that Grafana and **alertManagerMain** routes are not found in the cluster:

```
$ oc get route -n openshift-monitoring alertmanager-main
```

```
$ oc get route -n openshift-monitoring grafana
```

Both queries should return **Error from server (NotFound)** messages.

15. Check that there is a minimum of 4 CPUs allocated as **reserved** for each of the **PerformanceProfile, Tuned** performance-patch, workload partitioning, and kernel command line arguments by running the following command:

```
$ oc get performanceprofile -o jsonpath="{ .items[0].spec.cpu.reserved }"
```

Example output

```
0-1,52-53
```



NOTE

Depending on your workload requirements, you might require additional reserved CPUs to be allocated.

CHAPTER 22. WORKLOAD PARTITIONING ON SINGLE-NODE OPENSIFT

In resource-constrained environments, such as single-node OpenShift deployments, it is advantageous to reserve most of the CPU resources for your own workloads and configure OpenShift Container Platform to run on a fixed number of CPUs within the host. In these environments, management workloads, including the control plane, need to be configured to use fewer resources than they might by default in normal clusters. You can isolate the OpenShift Container Platform services, cluster management workloads, and infrastructure pods to run on a reserved set of CPUs.

When you use workload partitioning, the CPU resources used by OpenShift Container Platform for cluster management are isolated to a partitioned set of CPU resources on a single-node cluster. This partitioning isolates cluster management functions to the defined number of CPUs. All cluster management functions operate solely on that **cpuset** configuration.

The minimum number of reserved CPUs required for the management partition for a single-node cluster is four CPU Hyper threads (HTs). The set of pods that make up the baseline OpenShift Container Platform installation and a set of typical add-on Operators are annotated for inclusion in the management workload partition. These pods operate normally within the minimum size **cpuset** configuration. Inclusion of Operators or workloads outside of the set of accepted management pods requires additional CPU HTs to be added to that partition.

Workload partitioning isolates the user workloads away from the platform workloads using the normal scheduling capabilities of Kubernetes to manage the number of pods that can be placed onto those cores, and avoids mixing cluster management workloads and user workloads.

When applying workload partitioning, use the Node Tuning Operator to implement the performance profile:

- Workload partitioning pins the OpenShift Container Platform infrastructure pods to a defined **cpuset** configuration.
- The performance profile pins the systemd services to a defined **cpuset** configuration.
- This **cpuset** configuration must match.

Workload partitioning introduces a new extended resource of **<workload-type>.workload.openshift.io/cores** for each defined CPU pool, or workload-type. Kubelet advertises these new resources and CPU requests by pods allocated to the pool are accounted for within the corresponding resource rather than the typical **cpu** resource. When workload partitioning is enabled, the **<workload-type>.workload.openshift.io/cores** resource allows access to the CPU capacity of the host, not just the default CPU pool.

22.1. ENABLING WORKLOAD PARTITIONING

A key feature to enable as part of a single-node OpenShift installation is workload partitioning. This limits the cores allowed to run platform services, maximizing the CPU core for application payloads. You must configure workload partitioning at cluster installation time.



NOTE

You can enable workload partitioning during the cluster installation process only. You cannot disable workload partitioning post-installation. However, you can reconfigure workload partitioning by updating the **cpu** value that you define in the **performanceprofile**, and in the MachineConfig CR in the following procedure.

Procedure

- The base64-encoded content below contains the CPU set that the management workloads are constrained to. This content must be adjusted to match the set specified in the **performanceprofile** and must be accurate for the number of cores on the cluster.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 02-master-workload-partitioning
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,W2NyaW8ucnVudGltZS53b3JrbG9hZHMubWFuYWdlbWVudF0KYWN0aXZhdGlubl
9hbm5vdGF0aW9uID0gInRhcmlldC53b3JrbG9hZC5vcGVuc2hpZnQuaW8vbWVudGlubWVudC53b3JrbG9hZHMubWFuYWdlbWVudC5yZXNvdXJjZXNdCmNw
dXNoYXJlcjA9IDAKQ1BVcyA9IClwLTESIDUyLTUzIgo=
            mode: 420
            overwrite: true
            path: /etc/crio/crio.conf.d/01-workload-partitioning
            user:
              name: root
        - contents:
            source: data:text/plain;charset=utf-
8;base64,ewogICJtYW5hZ2VtZW50IjogewogICAgImNwdXNldCI6IClwLTESNTItNTMiCiAgfQp
9Cg==
            mode: 420
            overwrite: true
            path: /etc/kubernetes/openshift-workload-pinning
            user:
              name: root
```

- The contents of **/etc/crio/crio.conf.d/01-workload-partitioning** should look like this:

```
[crio.runtime.workloads.management]
activation_annotation = "target.workload.openshift.io/management"
annotation_prefix = "resources.workload.openshift.io"
[crio.runtime.workloads.management.resources]
cpushares = 0
CPUs = "0-1, 52-53" ❶
```

- ❶ The **CPUs** value varies based on the installation.

If Hyper-Threading is enabled, specify both threads of each core. The **CPUs** value must match the reserved CPU set specified in the performance profile.

This content should be base64 encoded and provided in the **01-workload-partitioning-content** in the manifest above.

- The contents of **/etc/kubernetes/openshift-workload-pinning** should look like this:

```
{  
  "management": {  
    "cpuset": "0-1,52-53" 1  
  }  
}
```

- 1 The **cpuset** must match the **CPUs** value in **/etc/crio/crio.conf.d/01-workload-partitioning**.

This content should be base64 encoded and provided in the **openshift-workload-pinning-content** in the preceding manifest.

CHAPTER 23. DEPLOYING DISTRIBUTED UNITS AT SCALE IN A DISCONNECTED ENVIRONMENT

Use zero touch provisioning (ZTP) to provision distributed units at new edge sites in a disconnected environment. The workflow starts when the site is connected to the network and ends with the CNF workload deployed and running on the site nodes.

23.1. PROVISIONING EDGE SITES AT SCALE

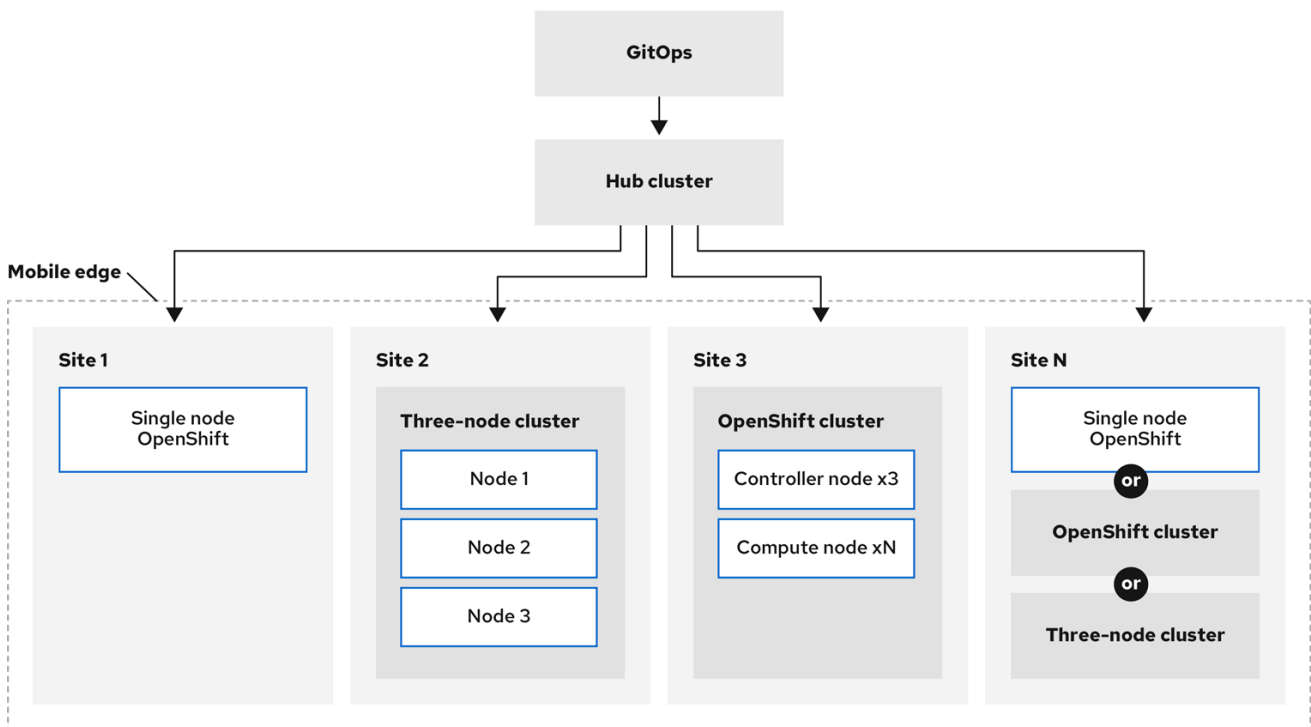
Telco edge computing presents extraordinary challenges with managing hundreds to tens of thousands of clusters in hundreds of thousands of locations. These challenges require fully-automated management solutions with, as closely as possible, zero human interaction.

Zero touch provisioning (ZTP) allows you to provision new edge sites with declarative configurations of bare-metal equipment at remote sites. Template or overlay configurations install OpenShift Container Platform features that are required for CNF workloads. End-to-end functional test suites are used to verify CNF related features. All configurations are declarative in nature.

You start the workflow by creating declarative configurations for ISO images that are delivered to the edge nodes to begin the installation process. The images are used to repeatedly provision large numbers of nodes efficiently and quickly, allowing you keep up with requirements from the field for far edge nodes.

Service providers are deploying a more distributed mobile network architecture allowed by the modular functional framework defined for 5G. This allows service providers to move from appliance-based radio access networks (RAN) to open cloud RAN architecture, gaining flexibility and agility in delivering services to end users.

The following diagram shows how ZTP works within a far edge framework.



217_OpenShift_0222

23.2. ABOUT ZTP AND DISTRIBUTED UNITS ON OPENSIFT CLUSTERS

You can install a distributed unit (DU) on OpenShift Container Platform clusters at scale with Red Hat Advanced Cluster Management (RHACM) using the assisted installer (AI) and the policy generator with core-reduction technology enabled. The DU installation is done using zero touch provisioning (ZTP) in a disconnected environment.

RHACM manages clusters in a hub-and-spoke architecture, where a single hub cluster manages many spoke clusters. RHACM applies radio access network (RAN) policies from predefined custom resources (CRs). Hub clusters running ACM provision and deploy the spoke clusters using ZTP and AI. DU installation follows the AI installation of OpenShift Container Platform on each cluster.

The AI service handles provisioning of OpenShift Container Platform on single node clusters, three-node clusters, or standard clusters running on bare metal. ACM ships with and deploys the AI when the **MultiClusterHub** custom resource is installed.

With ZTP and AI, you can provision OpenShift Container Platform clusters to run your DUs at scale. A high-level overview of ZTP for distributed units in a disconnected environment is as follows:

- A hub cluster running Red Hat Advanced Cluster Management (RHACM) manages a disconnected internal registry that mirrors the OpenShift Container Platform release images. The internal registry is used to provision the spoke clusters.
- You manage the bare metal host machines for your DUs in an inventory file that uses YAML for formatting. You store the inventory file in a Git repository.
- You install the DU bare metal host machines on site, and make the hosts ready for provisioning. To be ready for provisioning, the following is required for each bare metal host:
 - Network connectivity - including DNS for your network. Hosts should be reachable through the hub and managed spoke clusters. Ensure there is layer 3 connectivity between the hub and the host where you want to install your hub cluster.
 - Baseboard Management Controller (BMC) details for each host - ZTP uses BMC details to connect the URL and credentials for accessing the BMC. ZTP manages the spoke cluster definition CRs, with the exception of the **BMCSecret** CR, which you create manually. These define the relevant elements for the managed clusters.

23.3. THE GITOPS APPROACH

ZTP uses the GitOps deployment set of practices for infrastructure deployment that allows developers to perform tasks that would otherwise fall under the purview of IT operations. GitOps achieves these tasks using declarative specifications stored in Git repositories, such as YAML files and other defined patterns, that provide a framework for deploying the infrastructure. The declarative output is leveraged by the Open Cluster Manager (OCM) for multisite deployment.

One of the motivators for a GitOps approach is the requirement for reliability at scale. This is a significant challenge that GitOps helps solve.

GitOps addresses the reliability issue by providing traceability, RBAC, and a single source of truth for the desired state of each site. Scale issues are addressed by GitOps providing structure, tooling, and event driven operations through webhooks.

23.4. ZERO TOUCH PROVISIONING BUILDING BLOCKS

Red Hat Advanced Cluster Management (RHACM) leverages zero touch provisioning (ZTP) to deploy single-node OpenShift Container Platform clusters, three-node clusters, and standard clusters. The initial site plan is divided into smaller components and initial configuration data is stored in a Git repository. ZTP uses a declarative GitOps approach to deploy these clusters.

The deployment of the clusters includes:

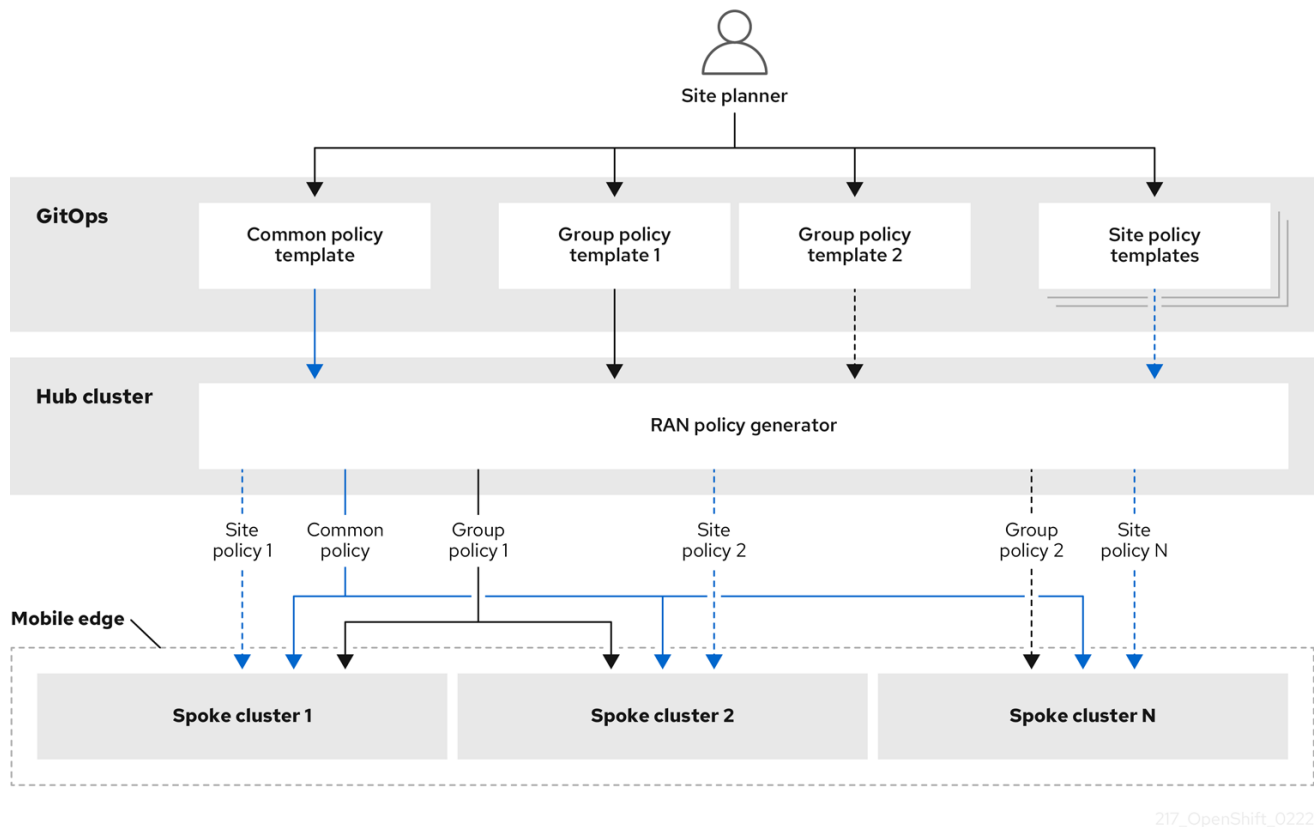
- Installing the host operating system (RHCOS) on a blank server.
- Deploying OpenShift Container Platform.
- Creating cluster policies and site subscriptions.
- Leveraging a GitOps deployment topology for a develop once, deploy anywhere model.
- Making the necessary network configurations to the server operating system.
- Deploying profile Operators and performing any needed software-related configuration, such as performance profile, PTP, and SR-IOV.
- Downloading images needed to run workloads (CNFs).

23.5. HOW TO PLAN YOUR RAN POLICIES

Zero touch provisioning (ZTP) uses Red Hat Advanced Cluster Management (RHACM) to apply the radio access network (RAN) configuration using a policy-based governance approach to apply the configuration.

The policy generator or **PolicyGen** is a part of the GitOps ZTP tooling that facilitates creating RHACM policies from a set of predefined custom resources. There are three main items: policy categorization, source CR policy, and the **PolicyGenTemplate** CR. **PolicyGen** uses these to generate the policies and their placement bindings and rules.

The following diagram shows how the RAN policy generator interacts with GitOps and RHACM.



217_OpenShift_0222

RAN policies are categorized into three main groups:

Common

A policy that exists in the **Common** category is applied to all clusters to be represented by the site plan. Cluster types include single node, three-node, and standard clusters.

Groups

A policy that exists in the **Groups** category is applied to a group of clusters. Every group of clusters could have their own policies that exist under the **Groups** category. For example, **Groups/group1** can have its own policies that are applied to the clusters belonging to **group1**. You can also define a group for each cluster type: single node, three-node, and standard clusters.

Sites

A policy that exists in the **Sites** category is applied to a specific cluster. Any cluster could have its own policies that exist in the **Sites** category. For example, **Sites/cluster1** has its own policies applied to **cluster1**. You can also define an example site-specific configuration for each cluster type: single node, three-node, and standard clusters.

23.6. LOW LATENCY FOR DISTRIBUTED UNITS (DUS)

Low latency is an integral part of the development of 5G networks. Telecommunications networks require as little signal delay as possible to ensure quality of service in a variety of critical use cases.

Low latency processing is essential for any communication with timing constraints that affect functionality and security. For example, 5G Telco applications require a guaranteed one millisecond one-way latency to meet Internet of Things (IoT) requirements. Low latency is also critical for the future development of autonomous vehicles, smart factories, and online gaming. Networks in these environments require almost a real-time flow of data.

Low latency systems are about guarantees with regards to response and processing times. This includes keeping a communication protocol running smoothly, ensuring device security with fast responses to

error conditions, or just making sure a system is not lagging behind when receiving a lot of data. Low latency is key for optimal synchronization of radio transmissions.

OpenShift Container Platform enables low latency processing for DUs running on COTS hardware by using a number of technologies and specialized hardware devices:

Real-time kernel for RHCOS

Ensures workloads are handled with a high degree of process determinism.

CPU isolation

Avoids CPU scheduling delays and ensures CPU capacity is available consistently.

NUMA awareness

Aligns memory and huge pages with CPU and PCI devices to pin guaranteed container memory and huge pages to the NUMA node. This decreases latency and improves performance of the node.

Huge pages memory management

Using huge page sizes improves system performance by reducing the amount of system resources required to access page tables.

Precision timing synchronization using PTP

Allows synchronization between nodes in the network with sub-microsecond accuracy.

23.7. PREPARING THE DISCONNECTED ENVIRONMENT

Before you can provision distributed units (DU) at scale, you must install Red Hat Advanced Cluster Management (RHACM), which handles the provisioning of the DUs.

RHACM is deployed as an Operator on the OpenShift Container Platform hub cluster. It controls clusters and applications from a single console with built-in security policies. RHACM provisions and manage your DU hosts. To install RHACM in a disconnected environment, you create a mirror registry that mirrors the Operator Lifecycle Manager (OLM) catalog that contains the required Operator images. OLM manages, installs, and upgrades Operators and their dependencies in the cluster.

You also use a disconnected mirror host to serve the RHCOS ISO and RootFS disk images that provision the DU bare-metal host operating system.

Additional resources

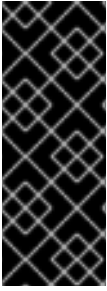
- For more information about creating the disconnected mirror registry, see [Creating a mirror registry](#).
- For more information about mirroring OpenShift Platform image to the disconnected registry, see [Mirroring images for a disconnected installation](#).

23.7.1. Adding RHCOS ISO and RootFS images to the disconnected mirror host

Before you install a cluster on infrastructure that you provision, you must create Red Hat Enterprise Linux CoreOS (RHCOS) machines for it to use. Use a disconnected mirror to host the RHCOS images you require to provision your distributed unit (DU) bare-metal hosts.

Prerequisites

- Deploy and configure an HTTP server to host the RHCOS image resources on the network. You must be able to access the HTTP server from your computer, and from the machines that you create.



IMPORTANT

The RHCOS images might not change with every release of OpenShift Container Platform. You must download images with the highest version that is less than or equal to the OpenShift Container Platform version that you install. Use the image versions that match your OpenShift Container Platform version if they are available. You require ISO and RootFS images to install RHCOS on the DU hosts. RHCOS qcow2 images are not supported for this installation type.

Procedure

1. Log in to the mirror host.
2. Obtain the RHCOS ISO and RootFS images from mirror.openshift.com, for example:
 - a. Export the required image names and OpenShift Container Platform version as environment variables:

```
$ export ISO_IMAGE_NAME=<iso_image_name> ❶
```

```
$ export ROOTFS_IMAGE_NAME=<rootfs_image_name> ❶
```

```
$ export OCP_VERSION=<ocp_version> ❶
```

❶ ISO image name, for example, **rhcos-4.11.0-fc.1-x86_64-live.x86_64.iso**

❶ RootFS image name, for example, **rhcos-4.11.0-fc.1-x86_64-live-rootfs.x86_64.img**

❶ OpenShift Container Platform version, for example, **latest-4.11**

- b. Download the required images:

```
$ sudo wget https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/pre-release/${OCP_VERSION}/${ISO_IMAGE_NAME} -O /var/www/html/${ISO_IMAGE_NAME}
```

```
$ sudo wget https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/pre-release/${OCP_VERSION}/${ROOTFS_IMAGE_NAME} -O /var/www/html/${ROOTFS_IMAGE_NAME}
```

Verification steps

- Verify that the images downloaded successfully and are being served on the disconnected mirror host, for example:

```
$ wget http://$(hostname)/${ISO_IMAGE_NAME}
```

Expected output

```
...
Saving to: rhcos-4.11.0-fc.1-x86_64-live.x86_64.iso
rhcos-4.11.0-fc.1-x86_64- 11%[====> ] 10.01M 4.71MB/s
```



23.8. INSTALLING RED HAT ADVANCED CLUSTER MANAGEMENT IN A DISCONNECTED ENVIRONMENT

You use Red Hat Advanced Cluster Management (RHACM) on a hub cluster in the disconnected environment to manage the deployment of distributed unit (DU) profiles on multiple managed spoke clusters.

Prerequisites

- Install the OpenShift Container Platform CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Configure a disconnected mirror registry for use in the cluster.



NOTE

If you want to deploy Operators to the spoke clusters, you must also add them to this registry. See [Mirroring an Operator catalog](#) for more information.

Procedure

- Install RHACM on the hub cluster in the disconnected environment. See [Installing RHACM in a disconnected environment](#).

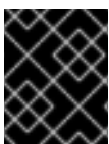
23.9. ENABLING ASSISTED INSTALLER SERVICE ON BARE METAL

The Assisted Installer Service (AIS) deploys OpenShift Container Platform clusters. Red Hat Advanced Cluster Management (RHACM) ships with AIS. AIS is deployed when you enable the MultiClusterHub Operator on the RHACM hub cluster.

For distributed units (DUs), RHACM supports OpenShift Container Platform deployments that run on a single bare-metal host, three-node clusters, or standard clusters. In the case of single node clusters or three-node clusters, all nodes act as both control plane and worker nodes.

Prerequisites

- Install OpenShift Container Platform 4.11 on a hub cluster.
- Install RHACM and create the **MultiClusterHub** resource.
- Create persistent volume custom resources (CR) for database and file system storage.
- You have installed the OpenShift CLI (**oc**).



IMPORTANT

Create a persistent volume resource for image storage. Failure to specify persistent volume storage for images can affect cluster performance.

Procedure

1. Modify the **Provisioning** resource to allow the Bare Metal Operator to watch all namespaces:

```
$ oc patch provisioning provisioning-configuration --type merge -p '{"spec":
{"watchAllNamespaces": true }}'
```

2. Create the **AgentServiceConfig** CR.

- a. Save the following YAML in the **agent_service_config.yaml** file:

```
apiVersion: agent-install.openshift.io/v1beta1
kind: AgentServiceConfig
metadata:
  name: agent
spec:
  databaseStorage:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <database_volume_size> 1
  filesystemStorage:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <file_storage_volume_size> 2
  imageStorage:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <image_storage_volume_size> 3
  osImages: 4
    - openshiftVersion: "<ocp_version>" 5
      version: "<ocp_release_version>" 6
      url: "<iso_url>" 7
      cpuArchitecture: "x86_64"
```

- 1 Volume size for the **databaseStorage** field, for example **10Gi**.
- 2 Volume size for the **filesystemStorage** field, for example **20Gi**.
- 3 Volume size for the **imageStorage** field, for example **2Gi**.
- 4 List of OS image details, for example a single OpenShift Container Platform OS version.
- 5 OpenShift Container Platform version to install, in either "x.y" (major.minor) or "x.y.z" (major.minor.patch) formats.
- 6 Specific install version, for example, **47.83.202103251640-0**.
- 7 ISO url, for example, https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/4.7/4.7.7/rhcos-4.7.7-x86_64-live.x86_64.iso.

- b. Create the **AgentServiceConfig** CR by running the following command:

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

Example output

```
agentserviceconfig.agent-install.openshift.io/agent created
```

23.10. ZTP CUSTOM RESOURCES

Zero touch provisioning (ZTP) uses custom resource (CR) objects to extend the Kubernetes API or introduce your own API into a project or a cluster. These CRs contain the site-specific data required to install and configure a cluster for RAN applications.

A custom resource definition (CRD) file defines your own object kinds. Deploying a CRD into the managed cluster causes the Kubernetes API server to begin serving the specified CR for the entire lifecycle.

For each CR in the **<site>.yaml** file on the managed cluster, ZTP uses the data to create installation CRs in a directory named for the cluster.

ZTP provides two ways for defining and installing CRs on managed clusters: a manual approach when you are provisioning a single cluster and an automated approach when provisioning multiple clusters.

Manual CR creation for single clusters

Use this method when you are creating CRs for a single cluster. This is a good way to test your CRs before deploying on a larger scale.

Automated CR creation for multiple managed clusters

Use the automated SiteConfig method when you are installing multiple managed clusters, for example, in batches of up to 100 clusters. SiteConfig uses ArgoCD as the engine for the GitOps method of site deployment. After completing a site plan that contains all of the required parameters for deployment, a policy generator creates the manifests and applies them to the hub cluster.

Both methods create the CRs shown in the following table. On the cluster site, an automated Discovery image ISO file creates a directory with the site name and a file with the cluster name. Every cluster has its own namespace, and all of the CRs are under that namespace. The namespace and the CR names match the cluster name.

Resource	Description	Usage
BareMetalHost	Contains the connection information for the Baseboard Management Controller (BMC) of the target bare-metal host.	Provides access to the BMC in order to load and boot the Discovery image ISO on the target server by using the Redfish protocol.
InfraEnv	Contains information for pulling OpenShift Container Platform onto the target bare-metal host.	Used with ClusterDeployment to generate the Discovery ISO for the managed cluster.

Resource	Description	Usage
AgentClusterInstall	Specifies the managed cluster's configuration such as networking and the number of supervisor (control plane) nodes. Shows the kubeconfig and credentials when the installation is complete.	Specifies the managed cluster configuration information and provides status during the installation of the cluster.
ClusterDeployment	References the AgentClusterInstall to use.	Used with InfraEnv to generate the Discovery ISO for the managed cluster.
NMStateConfig	Provides network configuration information such as MAC to IP mapping, DNS server, default route, and other network settings. This is not needed if DHCP is used.	Sets up a static IP address for the managed cluster's Kube API server.
Agent	Contains hardware information about the target bare-metal host.	Created automatically on the hub when the target machine's Discovery image ISO boots.
ManagedCluster	When a cluster is managed by the hub, it must be imported and known. This Kubernetes object provides that interface.	The hub uses this resource to manage and show the status of managed clusters.
KlusterletAddonConfig	Contains the list of services provided by the hub to be deployed to a ManagedCluster .	Tells the hub which add-on services to deploy to a ManagedCluster .
Namespace	Logical space for ManagedCluster resources existing on the hub. Unique per site.	Propagates resources to the ManagedCluster .
Secret	Two custom resources are created: BMC Secret and Image Pull Secret .	<ul style="list-style-type: none"> • BMC Secret authenticates into the target bare-metal host using its username and password. • Image Pull Secret contains authentication information for the OpenShift Container Platform image installed on the target bare-metal host.

Resource	Description	Usage
ClusterImageSet	Contains OpenShift Container Platform image information such as the repository and image name.	Passed into resources to provide OpenShift Container Platform images.

ZTP support for single node clusters, three-node clusters, and standard clusters requires updates to these CRs, including multiple instantiations of some.

ZTP provides support for deploying single node clusters, three-node clusters, and standard OpenShift clusters. This includes the installation of OpenShift and deployment of the distributed units (DUs) at scale.

The overall flow is identical to the ZTP support for single node clusters, with some differences in configuration depending on the type of cluster:

SiteConfig file:

- For single node clusters, the **SiteConfig** file must have exactly one entry in the **nodes** section.
- For three-node clusters, the **SiteConfig** file must have exactly three entries defined in the **nodes** section.
- For standard clusters, the **SiteConfig** file must have exactly three entries in the **nodes** section with **role: master** and one or more additional entries with **role: worker**.

PolicyGenTemplate file:

- The example common **PolicyGenTemplate** file is common across all types of clusters.
- There are example group **PolicyGenTemplate** files for single node, three-node, and standard clusters.
- Site-specific **PolicyGenTemplate** files are still specific to each site.

23.11. POLICYGENTEMPLATE CRS FOR RAN DEPLOYMENTS

You use **PolicyGenTemplate** custom resources (CRs) to customize the configuration applied to the cluster using the GitOps zero touch provisioning (ZTP) pipeline. The baseline configuration, obtained from the GitOps ZTP container, is designed to provide a set of critical features and node tuning settings that ensure the cluster can support the stringent performance and resource utilization constraints typical of RAN Distributed Unit (DU) applications. Changes or omissions from the baseline configuration can affect feature availability, performance, and resource utilization. Use **PolicyGenTemplate** CRs as the basis to create a hierarchy of configuration files tailored to your specific site requirements.

The baseline **PolicyGenTemplate** CRs that are defined for RAN DU cluster configuration can be extracted from the GitOps ZTP **ztp-site-generate**. See "Preparing the ZTP Git repository" for further details.

The **PolicyGenTemplate** CRs can be found in the `./out/argocd/example/policygentemplates` folder. The reference architecture has common, group, and site-specific configuration CRs. Each **PolicyGenTemplate** CR refers to other CRs that can be found in the `./out/source-crs` folder.

The **PolicyGenTemplate** CRs relevant to RAN cluster configuration are described below. Variants are provided for the group **PolicyGenTemplate** CRs to account for differences in single-node, three-node compact, and standard cluster configurations. Similarly, site-specific configuration variants are provided for single-node clusters and multi-node (compact or standard) clusters. Use the group and site-specific configuration variants that are relevant for your deployment.

Table 23.1. PolicyGenTemplate CRs for RAN deployments

PolicyGenTemplate CR	Description
common-ranGen.yaml	Contains a set of common RAN CRs that get applied to all clusters. These CRs subscribe to a set of operators providing cluster features typical for RAN as well as baseline cluster tuning.
group-du-3node-ranGen.yaml	Contains the RAN policies for three-node clusters only.
group-du-sno-ranGen.yaml	Contains the RAN policies for single-node clusters only.
group-du-standard-ranGen.yaml	Contains the RAN policies for standard three control-plane clusters.

Additional resources

- For more information about extracting the **/argocd** directory from the **ztp-site-generate** container image, see [Preparing the ZTP Git repository](#).

23.12. ABOUT THE POLICYGENTEMPLATE

The **PolicyGenTemplate.yaml** file is a custom resource definition (CRD) that tells the **PolicyGen** policy generator what CRs to include in the configuration, how to categorize the CRs into the generated policies, and what items in those CRs need to be updated with overlay content.

The following example shows a **PolicyGenTemplate.yaml** file:

```
---
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-du-sno"
  namespace: "group-du-sno-policies"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  sourceFiles:
    - fileName: ConsoleOperatorDisable.yaml
      policyName: "console-policy"
    - fileName: ClusterLogForwarder.yaml
      policyName: "log-forwarder-policy"
  spec:
```

```

outputs:
  - type: "kafka"
    name: kafka-open
    # below url is an example
    url: tcp://10.46.55.190:9092/test
pipelines:
  - name: audit-logs
    inputRefs:
      - audit
    outputRefs:
      - kafka-open
  - name: infrastructure-logs
    inputRefs:
      - infrastructure
    outputRefs:
      - kafka-open
- fileName: ClusterLogging.yaml
  policyName: "log-policy"
  spec:
    curation:
      curator:
        schedule: "30 3 * * *"
    collection:
      logs:
        type: "fluentd"
        fluentd: {}
- fileName: MachineConfigSctp.yaml
  policyName: "mc-sctp-policy"
  metadata:
    labels:
      machineconfiguration.openshift.io/role: master
- fileName: PtpConfigSlave.yaml
  policyName: "ptp-config-policy"
  metadata:
    name: "du-ptp-slave"
  spec:
    profile:
      - name: "slave"
        interface: "ens5f0"
        ptp4lOpts: "-2 -s --summary_interval -4"
        phc2sysOpts: "-a -r -n 24"
- fileName: SrioOperatorConfig.yaml
  policyName: "srio-operconfig-policy"
  spec:
    disableDrain: true
- fileName: MachineConfigAcceleratedStartup.yaml
  policyName: "mc-accelerated-policy"
  metadata:
    name: 04-accelerated-container-startup-master
    labels:
      machineconfiguration.openshift.io/role: master
- fileName: DisableSnoNetworkDiag.yaml
  policyName: "disable-network-diag"
  metadata:
    labels:
      machineconfiguration.openshift.io/role: master

```


The **group-du-ranGen.yaml** file defines a group of policies under a group named **group-du**. A Red Hat Advanced Cluster Management (RHACM) policy is generated for every source file that exists in **sourceFiles**. And, a single placement binding and placement rule is generated to apply the cluster selection rule for **group-du** policies.

Using the source file **PtpConfigSlave.yaml** as an example, the **PtpConfigSlave** has a definition of a **PtpConfig** custom resource (CR). The generated policy for the **PtpConfigSlave** example is named **group-du-ptp-config-policy**. The **PtpConfig** CR defined in the generated **group-du-ptp-config-policy** is named **du-ptp-slave**. The **spec** defined in **PtpConfigSlave.yaml** is placed under **du-ptp-slave** along with the other **spec** items defined under the source file.

The following example shows the **group-du-ptp-config-policy**:

```
apiVersion: policy.open-cluster-management.io/v1
kind: Policy
metadata:
  name: group-du-ptp-config-policy
  namespace: groups-sub
  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/standards: NIST SP 800-53
spec:
  remediationAction: enforce
  disabled: false
  policy-templates:
    - objectDefinition:
        apiVersion: policy.open-cluster-management.io/v1
        kind: ConfigurationPolicy
        metadata:
          name: group-du-ptp-config-policy-config
        spec:
          remediationAction: enforce
          severity: low
          namespaceselector:
            exclude:
              - kube-*
            include:
              - '*'
        object-templates:
          - complianceType: musthave
            objectDefinition:
              apiVersion: ptp.openshift.io/v1
              kind: PtpConfig
              metadata:
                name: slave
                namespace: openshift-ptp
              spec:
                recommend:
                  - match:
                      - nodeLabel: node-role.kubernetes.io/worker-du
                      priority: 4
                      profile: slave
                profile:
                  - interface: ens5f0
                    name: slave
```

```

phc2sysOpts: -a -r -n 24
ptp4lConf: |
  [global]
  #
  # Default Data Set
  #
  twoStepFlag 1
  slaveOnly 0
  priority1 128
  priority2 128
  domainNumber 24
  .....

```

23.13. BEST PRACTICES WHEN CUSTOMIZING POLICYGENTEMPLATE CRS

Consider the following best practices when customizing site configuration **PolicyGenTemplate** CRs:

- Use as few policies as necessary. Using fewer policies means using less resources. Each additional policy creates overhead for the hub cluster and the deployed spoke cluster. CRs are combined into policies based on the **policyName** field in the **PolicyGenTemplate** CR. CRs in the same **PolicyGenTemplate** which have the same value for **policyName** are managed under a single policy.
- Use a single catalog source for all Operators. In disconnected environments, configure the registry as a single index containing all Operators. Each additional **CatalogSource** on the spoke clusters increases CPU usage.
- **MachineConfig** CRs should be included as **extraManifests** in the **SiteConfig** CR so that they are applied during installation. This can reduce the overall time taken until the cluster is ready to deploy applications.
- **PolicyGenTemplates** should override the channel field to explicitly identify the desired version. This ensures that changes in the source CR during upgrades does not update the generated subscription.

Additional resources

- For details about best practice for scaling clusters with Red Hat Advanced Cluster Management (RHACM), see [ACM performance and scalability considerations](#).



NOTE

Scaling the hub cluster to managing large numbers of spoke clusters is affected by the number of policies created on the hub cluster. Grouping multiple configuration CRs into a single or limited number of policies is one way to reduce the overall number of policies on the hub cluster. When using the common/group/site hierarchy of policies for managing site configuration, it is especially important to combine site-specific configuration into a single policy.

23.14. CREATING THE POLICYGENTEMPLATE CR

Use this procedure to create the **PolicyGenTemplate** custom resource (CR) for your site in your local clone of the Git repository.

Prerequisites

Ensure that policy namespaces meets the following requirements:

- Namespace names must be prefixed with **ztp**. For example:

```
apiVersion: v1
kind: Namespace
metadata:
  name: ztp-common
```

- Namespaces must not match the namespace of a pre-existing cluster.

Procedure

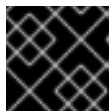
1. Choose an appropriate example from **out/argocd/example/policygentemplates**. This directory demonstrates a three-level policy framework that represents a well-supported low-latency profile tuned for the needs of 5G Telco DU deployments:
 - A single **common-ranGen.yaml** file that applies to all types of sites.
 - A set of shared **group-du-*-ranGen.yaml** files that are common between similar clusters.
 - An example **example-*-site.yaml** file that you can copy and update for each individual site.
2. Ensure that the labels defined in your **PolicyGenTemplate bindingRules** section correspond to the labels that are defined in the **SiteConfig** files of the clusters you are managing.
3. Ensure that the content of the overlaid spec files matches your desired end state. As a reference, the **out/source-crs** directory contains the full list of **source-crs** available to be included and overlaid by your **PolicyGenTemplate** templates.



NOTE

Depending on the specific requirements of your clusters, you might need more than a single group policy per cluster type, especially considering that the example group policies each have a single **PerformancePolicy.yaml** file that can only be shared across a set of clusters if those clusters consist of identical hardware configurations.

4. Define all the policy namespaces in a YAML file similar to the example **out/argocd/example/policygentemplates/ns.yaml** file.



IMPORTANT

Ensure that policy namespaces begin with **ztp** and are unique.

5. Add all the **PolicyGenTemplate** files and **ns.yaml** file to the **kustomization.yaml** file, similar to the example **out/argocd/example/policygentemplates/kustomization.yaml** file.
6. Commit the **PolicyGenTemplate** CRs, **ns.yaml** file, and the associated **kustomization.yaml** file in the Git repository.

23.15. CONFIGURING POLICY COMPLIANCE EVALUATION TIMEOUTS FOR POLICYGENTEMPLATE CRS

Use Red Hat Advanced Cluster Management (RHACM) installed on a hub cluster to monitor and report on whether your managed clusters are compliant with applied policies. RHACM uses policy templates to apply predefined policy controllers and policies. Policy controllers are Kubernetes custom resource definition (CRD) instances.

You can override the default policy evaluation intervals with **PolicyGenTemplate** custom resources (CRs). You configure duration settings that define how long a **ConfigurationPolicy** CR can be in a state of policy compliance or non-compliance before RHACM re-evaluates the applied cluster policies.

The zero touch provisioning (ZTP) policy generator generates **ConfigurationPolicy** CR policies with pre-defined policy evaluation intervals. The default value for the **noncompliant** state is 10 seconds. The default value for the **compliant** state is 10 minutes. To disable the evaluation interval, set the value to **never**.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in to the hub cluster as a user with **cluster-admin** privileges.
- You have created a Git repository where you manage your custom site configuration data.

Procedure

1. To configure the evaluation interval for all policies in a **PolicyGenTemplate** CR, add **evaluationInterval** to the **spec** field, and then set the appropriate **compliant** and **noncompliant** values. For example:

```
spec:
  evaluationInterval:
    compliant: 30m
    noncompliant: 20s
```

2. To configure the evaluation interval for the **spec.sourceFiles** object in a **PolicyGenTemplate** CR, add **evaluationInterval** to the **sourceFiles** field, for example:

```
spec:
  sourceFiles:
    - fileName: SriovSubscription.yaml
      policyName: "sriov-sub-policy"
      evaluationInterval:
        compliant: never
        noncompliant: 10s
```

3. Commit the **PolicyGenTemplate** CRs files in the Git repository and push your changes.

Verification

Check that the managed spoke cluster policies are monitored at the expected intervals.

1. Log in as a user with **cluster-admin** privileges on the managed cluster.

2. Get the pods that are running in the **open-cluster-management-agent-addon** namespace. Run the following command:

```
$ oc get pods -n open-cluster-management-agent-addon
```

Example output

```
NAME                                READY STATUS RESTARTS   AGE
config-policy-controller-858b894c68-v4xdb  1/1   Running  22 (5d8h ago)  10d
```

3. Check the applied policies are being evaluated at the expected interval in the logs for the **config-policy-controller** pod:

```
$ oc logs -n open-cluster-management-agent-addon config-policy-controller-858b894c68-v4xdb
```

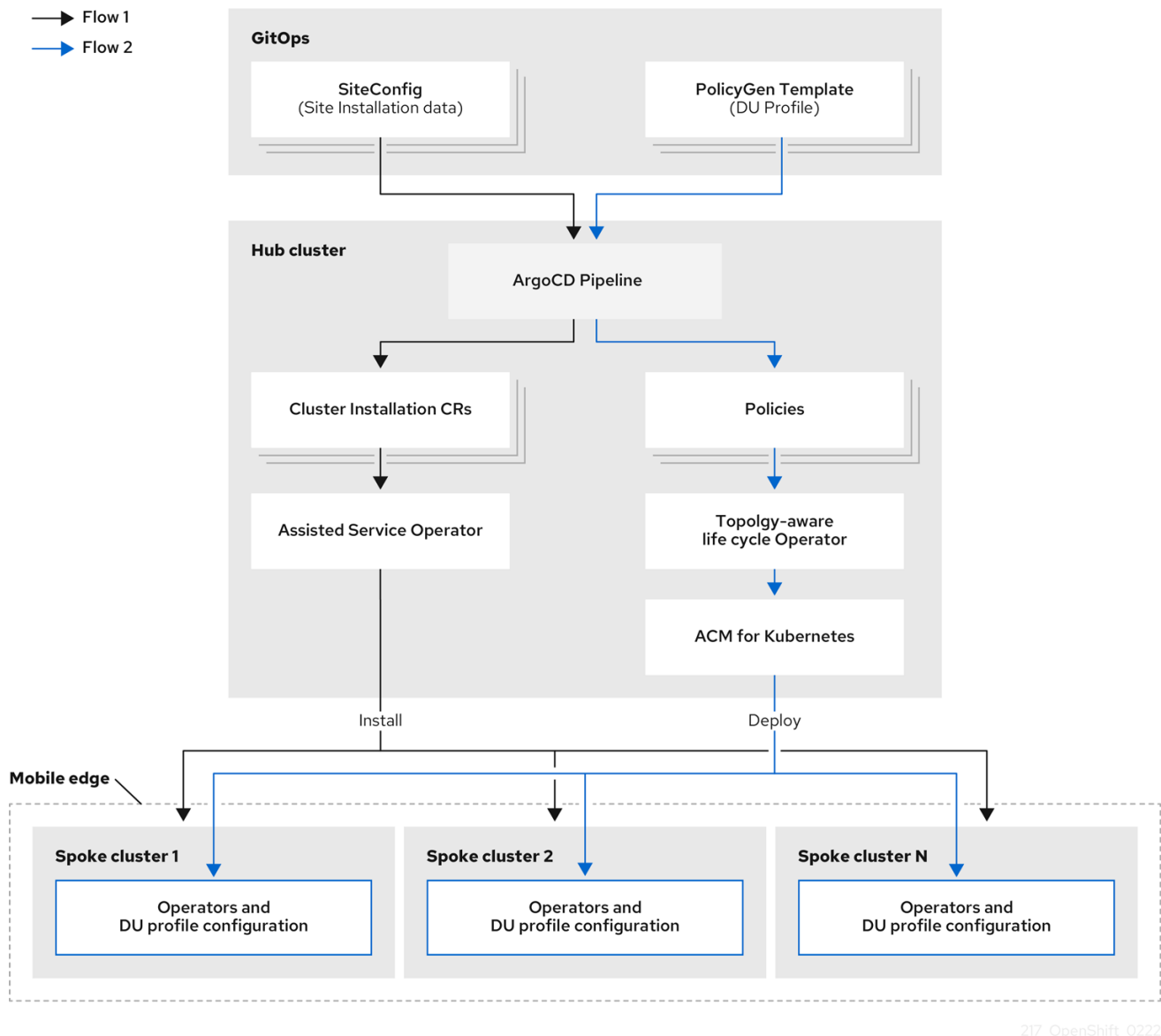
Example output

```
2022-05-10T15:10:25.280Z    info  configuration-policy-controller
controllers/configurationpolicy_controller.go:166    Skipping the policy evaluation due to the
policy not reaching the evaluation interval {"policy": "compute-1-config-policy-config"}
2022-05-10T15:10:25.280Z    info  configuration-policy-controller
controllers/configurationpolicy_controller.go:166    Skipping the policy evaluation due to the
policy not reaching the evaluation interval {"policy": "compute-1-common-compute-1-catalog-
policy-config"}
```

23.16. CREATING ZTP CUSTOM RESOURCES FOR MULTIPLE MANAGED CLUSTERS

If you are installing multiple managed clusters, zero touch provisioning (ZTP) uses ArgoCD and **SiteConfig** files to manage the processes that create the CRs and generate and apply the policies for multiple clusters, in batches of no more than 100, using the GitOps approach.

Installing and deploying the clusters is a two stage process, as shown here:



23.16.1. Using PolicyGenTemplate CRs to override source CRs content

PolicyGenTemplate CRs allow you to overlay additional configuration details on top of the base source CRs provided in the **ztp-site-generate** container. You can think of **PolicyGenTemplate** CRs as a logical merge or patch to the base CR. Use **PolicyGenTemplate** CRs to update a single field of the base CR, or overlay the entire contents of the base CR. You can update values and insert fields that are not in the base CR.

The following example procedure describes how to update fields in the generated **PerformanceProfile** CR for the reference configuration based on the **PolicyGenTemplate** CR in the **group-du-sno-ranGen.yaml** file. Use the procedure as a basis for modifying other parts of the **PolicyGenTemplate** based on your requirements.

Prerequisites

- Create a Git repository where you manage your custom site configuration data. The repository must be accessible from the hub cluster and be defined as a source repository for Argo CD.

Procedure

1. Review the baseline source CR for existing content. You can review the source CRs listed in the reference **PolicyGenTemplate** CRs by extracting them from the zero touch provisioning (ZTP) container.

- a. Create an **/out** folder:

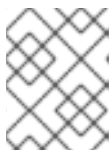
```
$ mkdir -p ./out
```

- b. Extract the source CRs:

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.11 extract /home/ztp --tar | tar x -C ./out
```

2. Review the baseline **PerformanceProfile** CR in **./out/source-crs/PerformanceProfile.yaml**:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: $name
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  additionalKernelArgs:
    - "idle=poll"
    - "rcupdate.rcu_normal_after_boot=0"
  cpu:
    isolated: $isolated
    reserved: $reserved
  hugepages:
    defaultHugepagesSize: $defaultHugepagesSize
  pages:
    - size: $size
      count: $count
      node: $node
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/$mcp: ""
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/$mcp: ""
  numa:
    topologyPolicy: "restricted"
  realTimeKernel:
    enabled: true
```



NOTE

Any fields in the source CR which contain **\$...** are removed from the generated CR if they are not provided in the **PolicyGenTemplate** CR.

3. Update the **PolicyGenTemplate** entry for **PerformanceProfile** in the **group-du-sno-ranGen.yaml** reference file. The following example **PolicyGenTemplate** CR stanza supplies appropriate CPU specifications, sets the **hugepages** configuration, and adds a new field that sets **globallyDisableIrqLoadBalancing** to false.

—

```

- fileName: PerformanceProfile.yaml
  policyName: "config-policy"
  metadata:
    name: openshift-node-performance-profile
  spec:
    cpu:
      # These must be tailored for the specific hardware platform
      isolated: "2-19,22-39"
      reserved: "0-1,20-21"
    hugepages:
      defaultHugepagesSize: 1G
    pages:
      - size: 1G
      count: 10
    globallyDisableIrqLoadBalancing: false

```

4. Commit the **PolicyGenTemplate** change in Git, and then push to the Git repository being monitored by the GitOps ZTP argo CD application.

Example output

The ZTP application generates an ACM policy that contains the generated **PerformanceProfile** CR. The contents of that CR are derived by merging the **metadata** and **spec** contents from the **PerformanceProfile** entry in the **PolicyGenTemplate** onto the source CR. The resulting CR has the following content:

```

---
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: openshift-node-performance-profile
spec:
  additionalKernelArgs:
    - idle=poll
    - rcupdate.rcu_normal_after_boot=0
  cpu:
    isolated: 2-19,22-39
    reserved: 0-1,20-21
  globallyDisableIrqLoadBalancing: false
  hugepages:
    defaultHugepagesSize: 1G
    pages:
      - count: 10
      size: 1G
  machineConfigPoolSelector:
    pools.operator.machineconfiguration.openshift.io/master: ""
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/master: ""
  numa:
    topologyPolicy: restricted
  realTimeKernel:
    enabled: true

```


NOTE

In the **/source-crs** folder that you extract from the **ztp-site-generate** container, the **\$** syntax is not used for template substitution as implied by the syntax. Rather, if the **policyGen** tool sees the **\$** prefix for a string and you do not specify a value for that field in the related **PolicyGenTemplate** CR, the field is omitted from the output CR entirely.

An exception to this is the **\$mcp** variable in **/source-crs** YAML files that is substituted with the specified value for **mcp** from the **PolicyGenTemplate** CR. For example, in **example/policygentemplates/group-du-standard-ranGen.yaml**, the value for **mcp** is **worker**:

```
spec:
  bindingRules:
    group-du-standard: ""
    mcp: "worker"
```

The **policyGen** tool replace instances of **\$mcp** with **worker** in the output CRs.

23.16.2. Filtering custom resources using SiteConfig filters

By using filters, you can easily customize **SiteConfig** custom resources (CRs) to include or exclude other CRs for use in the installation phase of the zero touch provisioning (ZTP) GitOps pipeline.

You can specify an **inclusionDefault** value of **include** or **exclude** for the **SiteConfig** CR, along with a list of the specific **extraManifest** RAN CRs that you want to include or exclude. Setting **inclusionDefault** to **include** makes the ZTP pipeline apply all the files in **/source-crs/extra-manifest** during installation. Setting **inclusionDefault** to **exclude** does the opposite.

You can exclude individual CRs from the **/source-crs/extra-manifest** folder that are otherwise included by default. The following example configures a custom single-node OpenShift **SiteConfig** CR to exclude the **/source-crs/extra-manifest/03-sctp-machine-config-worker.yaml** CR at installation time.

Some additional optional filtering scenarios are also described.

Prerequisites

- You configured the hub cluster for generating the required installation and policy CRs.
- You created a Git repository where you manage your custom site configuration data. The repository must be accessible from the hub cluster and be defined as a source repository for the Argo CD application.

Procedure

1. To prevent the ZTP pipeline from applying the **03-sctp-machine-config-worker.yaml** CR file, apply the following YAML in the **SiteConfig** CR:

```
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "site1-sno-du"
  namespace: "site1-sno-du"
spec:
  baseDomain: "example.com"
```

```

pullSecretRef:
  name: "assisted-deployment-pull-secret"
clusterImageSetNameRef: "openshift-4.11"
sshPublicKey: "<ssh_public_key>"
clusters:
- clusterName: "site1-sno-du"
extraManifests:
  filter:
    exclude:
      - 03-sctp-machine-config-worker.yaml

```

The ZTP pipeline skips the **03-sctp-machine-config-worker.yaml** CR during installation. All other CRs in **/source-crs/extra-manifest** are applied.

2. Save the **SiteConfig** CR and push the changes to the site configuration repository. The ZTP pipeline monitors and adjusts what CRs it applies based on the **SiteConfig** filter instructions.
3. Optional: To prevent the ZTP pipeline from applying all the **/source-crs/extra-manifest** CRs during cluster installation, apply the following YAML in the **SiteConfig** CR:

```

- clusterName: "site1-sno-du"
extraManifests:
  filter:
    inclusionDefault: exclude

```

4. Optional: To exclude all the **/source-crs/extra-manifest** RAN CRs and instead include a custom CR file during installation, edit the custom **SiteConfig** CR to set the custom manifests folder and the **include** file, for example:

```

clusters:
- clusterName: "site1-sno-du"
extraManifestPath: "<custom_manifest_folder>" ❶
extraManifests:
  filter:
    inclusionDefault: exclude ❷
    include:
      - custom-sctp-machine-config-worker.yaml

```

❶ Replace **<custom_manifest_folder>** with the name of the folder that contains the custom installation CRs, for example, **user-custom-manifest/**.

❷ Set **inclusionDefault** to **exclude** to prevent the ZTP pipeline from applying the files in **/source-crs/extra-manifest** during installation.

The following example illustrates the custom folder structure:

```

siteconfig
├── site1-sno-du.yaml
├── user-custom-manifest
│   └── custom-sctp-machine-config-worker.yaml

```

23.16.3. Configuring PTP fast events using PolicyGenTemplate CRs

You can configure PTP fast events for vRAN clusters that are deployed using the GitOps Zero Touch Provisioning (ZTP) pipeline. Use **PolicyGenTemplate** custom resources (CRs) as the basis to create a hierarchy of configuration files tailored to your specific site requirements.

Prerequisites

- Create a Git repository where you manage your custom site configuration data.

Procedure

1. Add the following YAML into **.spec.sourceFiles** in the **common-ranGen.yaml** file to configure the AMQP Operator:

```
#AMQ interconnect operator for fast events
- fileName: AmqSubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscription.yaml
  policyName: "subscriptions-policy"
```

2. Apply the following **PolicyGenTemplate** changes to **group-du-3node-ranGen.yaml**, **group-du-sno-ranGen.yaml**, or **group-du-standard-ranGen.yaml** files according to your requirements:

- a. In **.sourceFiles**, add the **PtpOperatorConfig** CR file that configures the AMQ transport host to the **config-policy**:

```
- fileName: PtpOperatorConfigForEvent.yaml
  policyName: "config-policy"
```

- b. Configure the **linuxptp** and **phc2sys** for the PTP clock type and interface. For example, add the following stanza into **.sourceFiles**:

```
- fileName: PtpConfigSlave.yaml ❶
  policyName: "config-policy"
  metadata:
    name: "du-ptp-slave"
  spec:
    profile:
      - name: "slave"
        interface: "ens5f1" ❷
        ptp4lOpts: "-2 -s --summary_interval -4" ❸
        phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" ❹
        ptpClockThreshold: ❺
        holdOverTimeout: 30 #secs
        maxOffsetThreshold: 100 #nano secs
        minOffsetThreshold: -100 #nano secs
```

- ❶ Can be one **PtpConfigMaster.yaml**, **PtpConfigSlave.yaml**, or **PtpConfigSlaveCvl.yaml** depending on your requirements. **PtpConfigSlaveCvl.yaml** configures **linuxptp** services for an Intel E810 Columbiaville NIC. For configurations based on **group-du-sno-ranGen.yaml** or **group-du-3node-ranGen.yaml**, use **PtpConfigSlave.yaml**.

- 2 Device specific interface name.
 - 3 You must append the `--summary_interval -4` value to `ptp4lOpts` in `.spec.sourceFiles.spec.profile` to enable PTP fast events.
 - 4 Required `phc2sysOpts` values. `-m` prints messages to `stdout`. The `linuxptp-daemon DaemonSet` parses the logs and generates Prometheus metrics.
 - 5 Optional. If the `ptpClockThreshold` stanza is not present, default values are used for the `ptpClockThreshold` fields. The stanza shows default `ptpClockThreshold` values. The `ptpClockThreshold` values configure how long after the PTP master clock is disconnected before PTP events are triggered. `holdOverTimeout` is the time value in seconds before the PTP clock event state changes to `FREERUN` when the PTP master clock is disconnected. The `maxOffsetThreshold` and `minOffsetThreshold` settings configure offset values in nanoseconds that compare against the values for `CLOCK_REALTIME (phc2sys)` or master offset (`ptp4l`). When the `ptp4l` or `phc2sys` offset value is outside this range, the PTP clock state is set to `FREERUN`. When the offset value is within this range, the PTP clock state is set to `LOCKED`.
3. Apply the following **PolicyGenTemplate** changes to your specific site YAML files, for example, **example-sno-site.yaml**:
 - a. In `.sourceFiles`, add the **Interconnect** CR file that configures the AMQ router to the **config-policy**:


```
- fileName: AmqInstance.yaml
  policyName: "config-policy"
```
 4. Merge any other required changes and files with your custom site repository.
 5. Push the changes to your site configuration repository to deploy PTP fast events to new sites using GitOps ZTP.

23.16.4. Configuring UEFI secure boot for clusters using PolicyGenTemplate CRs

You can configure UEFI secure boot for vRAN clusters that are deployed using the GitOps zero touch provisioning (ZTP) pipeline.

Prerequisites

- Create a Git repository where you manage your custom site configuration data.

Procedure

1. Create the following **MachineConfig** resource and save it in the **uefi-secure-boot.yaml** file:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: uefi-secure-boot
spec:
  config:
```

```

ignition:
  version: 3.1.0
kernelArguments:
  - efi=runtime

```

2. In your Git repository custom **/siteconfig** directory, create a **/sno-extra-manifest** folder and add the **uefi-secure-boot.yaml** file, for example:

```

siteconfig
├── site1-sno-du.yaml
├── site2-standard-du.yaml
├── sno-extra-manifest
│   └── uefi-secure-boot.yaml

```

3. In your cluster **SiteConfig** CR, specify the required values for **extraManifestPath** and **bootMode**:
 - a. Enter the directory name in the **.spec.clusters.extraManifestPath** field, for example:

```

clusters:
  - clusterName: "example-cluster"
    extraManifestPath: sno-extra-manifest/

```

- b. Set the value for **.spec.clusters.nodes.bootMode** to **UEFISecureBoot**, for example:

```

nodes:
  - hostName: "ran.example.lab"
    bootMode: "UEFISecureBoot"

```

4. Deploy the cluster using the GitOps ZTP pipeline.

Verification

1. Open a remote shell to the deployed cluster, for example:

```
$ oc debug node/node-1.example.com
```

2. Verify that the **SecureBoot** feature is enabled:

```
sh-4.4# mokutil --sb-state
```

Example output

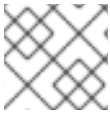
```
SecureBoot enabled
```

23.16.5. Configuring bare-metal event monitoring using PolicyGenTemplate CRs

You can configure bare-metal hardware events for vRAN clusters that are deployed using the GitOps Zero Touch Provisioning (ZTP) pipeline.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Create a Git repository where you manage your custom site configuration data.

**NOTE**

Multiple **HardwareEvent** resources are not permitted.

Procedure

1. To configure the AMQ Interconnect Operator and the Bare Metal Event Relay Operator, add the following YAML to **spec.sourceFiles** in the **common-ranGen.yaml** file:

```
# AMQ interconnect operator for fast events
- fileName: AmqSubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: AmqSubscription.yaml
  policyName: "subscriptions-policy"
# Bare Metal Event Relay operator
- fileName: BareMetalEventRelaySubscriptionNS.yaml
  policyName: "subscriptions-policy"
- fileName: BareMetalEventRelaySubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
- fileName: BareMetalEventRelaySubscription.yaml
  policyName: "subscriptions-policy"
```

2. Add the **Interconnect** CR to **.spec.sourceFiles** in the site configuration file, for example, the **example-sno-site.yaml** file:

```
- fileName: AmqInstance.yaml
  policyName: "config-policy"
```

3. Add the **HardwareEvent** CR to **spec.sourceFiles** in your specific group configuration file, for example, in the **group-du-sno-ranGen.yaml** file:

```
- fileName: HardwareEvent.yaml
  policyName: "config-policy"
  spec:
    nodeSelector: {}
    transportHost: "amqp://<amq_interconnect_name>.<amq_interconnect_namespace>.svc.cluster.local" 1
    logLevel: "info"
```

1

The **transportHost** URL is composed of the existing AMQ Interconnect CR **name** and **namespace**. For example, in **transportHost: "amqp://amq-router.amq-router.svc.cluster.local"**, the AMQ Interconnect **name** and **namespace** are both set to **amq-router**.

4. Commit the **PolicyGenTemplate** change in Git, and then push the changes to your site configuration repository to deploy bare-metal events monitoring to new sites using GitOps ZTP.
5. Create the Redfish Secret by running the following command:

```
$ oc -n openshift-bare-metal-events create secret generic redfish-basic-auth \
--from-literal=username=<bmc_username> --from-literal=password=<bmc_password> \
--from-literal=hostaddr="<bmc_host_ip_addr>"
```

Additional resources

- For more information about how to install the Bare Metal Event Relay, see [Installing the Bare Metal Event Relay using the CLI](#).
- For more information about how to install the AMQ Interconnect Operator, see [Installing the AMQ messaging bus](#).
- For more information about how to create the username, password, and the host IP address for the secret, see [Creating the bare-metal event and Secret CRs](#).

23.17. INSTALLING THE GITOPS ZTP PIPELINE

The procedures in this section tell you how to complete the following tasks:

- Prepare the Git repository you need to host site configuration data.
- Configure the hub cluster for generating the required installation and policy custom resources (CR).
- Deploy the managed clusters using zero touch provisioning (ZTP).

23.17.1. Preparing the ZTP Git repository

Create a Git repository for hosting site configuration data. The zero touch provisioning (ZTP) pipeline requires read access to this repository.

Procedure

1. Create a directory structure with separate paths for the **SiteConfig** and **PolicyGenTemplate** custom resources (CR).
2. Export the **argocd** directory from the **ztp-site-generate** container image using the following commands:

```
$ podman pull registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.11
```

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

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.11 extract /home/ztp --tar | tar x -C ./out
```

3. Check that the **out** directory contains the following subdirectories:

- **out/extra-manifest** contains the source CR files that **SiteConfig** uses to generate extra manifest **configMap**.
- **out/source-crs** contains the source CR files that **PolicyGenTemplate** uses to generate the Red Hat Advanced Cluster Management (RHACM) policies.
- **out/argocd/deployment** contains patches and YAML files to apply on the hub cluster for use in the next step of this procedure.
- **out/argocd/example** contains the examples for **SiteConfig** and **PolicyGenTemplate** files that represent the recommended configuration.

The directory structure under **out/argocd/example** serves as a reference for the structure and content of your Git repository. The example includes **SiteConfig** and **PolicyGenTemplate** reference CRs for single-node, three-node, and standard clusters. Remove references to cluster types that you are not using. The following example describes a set of CRs for a network of single-node clusters:

```
example/
├── policygentemplates
│   ├── common-ranGen.yaml
│   ├── example-sno-site.yaml
│   ├── group-du-sno-ranGen.yaml
│   ├── group-du-sno-validator-ranGen.yaml
│   ├── kustomization.yaml
│   └── ns.yaml
└── siteconfig
    ├── example-sno.yaml
    ├── KlusterletAddonConfigOverride.yaml
    └── kustomization.yaml
```

Keep **SiteConfig** and **PolicyGenTemplate** CRs in separate directories. Both the **SiteConfig** and **PolicyGenTemplate** directories must contain a **kustomization.yaml** file that explicitly includes the files in that directory.

This directory structure and the **kustomization.yaml** files must be committed and pushed to your Git repository. The initial push to Git should include the **kustomization.yaml** files. The **SiteConfig** (**example-sno.yaml**) and **PolicyGenTemplate** (**common-ranGen.yaml**, **group-du-sno*.yaml**, and **example-sno-site.yaml**) files can be omitted and pushed at a later time as required when deploying a site.

The **KlusterletAddonConfigOverride.yaml** file is only required if one or more **SiteConfig** CRs which make reference to it are committed and pushed to Git. See **example-sno.yaml** for an example of how this is used.

23.17.2. Preparing the hub cluster for ZTP

You can configure your hub cluster with a set of ArgoCD applications that generate the required installation and policy custom resources (CR) for each site based on a zero touch provisioning (ZTP) GitOps flow.

Prerequisites

- OpenShift Cluster 4.11 as the hub cluster
- Red Hat Advanced Cluster Management (RHACM) Operator 2.5 installed on the hub cluster

- Red Hat OpenShift GitOps Operator 1.5 on the hub cluster

Procedure

1. Install the Topology Aware Lifecycle Manager (TALM), which coordinates with any new sites added by ZTP and manages application of the **PolicyGenTemplate**-generated policies.
2. Prepare the ArgoCD pipeline configuration:
 - a. Create a Git repository with the directory structure similar to the example directory. For more information, see "Preparing the ZTP Git repository".
 - b. Configure access to the repository using the ArgoCD UI. Under **Settings** configure the following:
 - **Repositories** - Add the connection information. The URL must end in **.git**, for example, <https://repo.example.com/repo.git> and credentials.
 - **Certificates** - Add the public certificate for the repository, if needed.
 - c. Modify the two ArgoCD Applications, **out/argocd/deployment/clusters-app.yaml** and **out/argocd/deployment/policies-app.yaml**, based on your Git repository:
 - Update the URL to point to the Git repository. The URL must end with **.git**, for example, <https://repo.example.com/repo.git>.
 - The **targetRevision** must indicate which Git repository branch to monitor.
 - The path should specify the path to the **SiteConfig** or **PolicyGenTemplate** CRs, respectively.
3. To patch the ArgoCD instance in the hub cluster by using the patch file previously extracted into the **out/argocd/deployment/** directory, enter the following command:

```
$ oc patch argocd openshift-gitops \
-n openshift-gitops --type=merge \
--patch-file out/argocd/deployment/argocd-openshift-gitops-patch.json
```

4. Apply the pipeline configuration to your hub cluster by using the following command:

```
$ oc apply -k out/argocd/deployment
```

23.17.3. Deploying additional changes to clusters

Custom resources (CRs) that are deployed through the GitOps zero touch provisioning (ZTP) pipeline support two goals:

1. Deploying additional Operators to spoke clusters that are required by typical RAN DU applications running at the network far-edge.
2. Customizing the OpenShift Container Platform installation to provide a high performance platform capable of meeting the strict timing requirements in a minimal CPU budget.

If you require cluster configuration changes outside of the base GitOps ZTP pipeline configuration, there are three options:

Apply the additional configuration after the ZTP pipeline is complete

When the GitOps ZTP pipeline deployment is complete, the deployed cluster is ready for application workloads. At this point, you can install additional Operators and apply configurations specific to your requirements. Ensure that additional configurations do not negatively affect the performance of the platform or allocated CPU budget.

Add content to the ZTP library

The base source CRs that you deploy with the GitOps ZTP pipeline can be augmented with custom content as required.

Create extra manifests for the cluster installation

Extra manifests are applied during installation and makes the installation process more efficient.



IMPORTANT

Providing additional source CRs or modifying existing source CRs can significantly impact the performance or CPU profile of OpenShift Container Platform.

Additional resources

- See [Adding new content to the GitOps ZTP pipeline](#) for more information about adding or modifying existing source CRs in the **ztp-site-generate** container.
- See [Customizing the ZTP GitOps pipeline with extra manifests](#) for more information on adding extra manifests.

23.18. ADDING NEW CONTENT TO THE GITOPS ZTP PIPELINE

The source CRs in the GitOps ZTP site generator container provide a set of critical features and node tuning settings for RAN Distributed Unit (DU) applications. These are applied to the clusters that you deploy with ZTP. To add or modify existing source CRs in the **ztp-site-generate** container, rebuild the **ztp-site-generate** container and make it available to the hub cluster, typically from the disconnected registry associated with the hub cluster. Any valid OpenShift Container Platform CR can be added.

Perform the following procedure to add new content to the ZTP pipeline.

Procedure

1. Create a directory containing a Containerfile and the source CR YAML files that you want to include in the updated **ztp-site-generate** container, for example:

```
ztp-update/
├── example-cr1.yaml
├── example-cr2.yaml
└── ztp-update.in
```

2. Add the following content to the **ztp-update.in** Containerfile:

```
FROM registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.11

ADD example-cr2.yaml /kustomize/plugin/ran.openshift.io/v1/policygentemplate/source-crs/
ADD example-cr1.yaml /kustomize/plugin/ran.openshift.io/v1/policygentemplate/source-crs/
```

3. Open a terminal at the **ztp-update/** folder and rebuild the container:

```
$ podman build -t ztp-site-generate-rhel8-custom:v4.11-custom-1
```

4. Push the built container image to your disconnected registry, for example:

```
$ podman push localhost/ztp-site-generate-rhel8-custom:v4.11-custom-1
registry.example.com:5000/ztp-site-generate-rhel8-custom:v4.11-custom-1
```

5. Patch the Argo CD instance on the hub cluster to point to the newly built container image:

```
$ oc patch -n openshift-gitops argocd openshift-gitops --type=json -p '[{"op": "replace",
"path": "/spec/repo/initContainers/0/image", "value": "registry.example.com:5000/ztp-site-
generate-rhel8-custom:v4.11-custom-1"} ]'
```

When the Argo CD instance is patched, the **openshift-gitops-repo-server** pod automatically restarts.

Verification

1. Verify that the new **openshift-gitops-repo-server** pod has completed initialization and that the previous repo pod is terminated:

```
$ oc get pods -n openshift-gitops | grep openshift-gitops-repo-server
```

Example output

```
openshift-gitops-server-7df86f9774-db682      1/1    Running      1      28s
```

You must wait until the new **openshift-gitops-repo-server** pod has completed initialization and the previous pod is terminated before the newly added container image content is available.

Additional resources

- Alternatively, you can patch the Argo CD instance as described in [Preparing the hub cluster for ZTP](#) by modifying **argocd-openshift-gitops-patch.json** with an updated **initContainer** image before applying the patch file.

23.19. CUSTOMIZING EXTRA INSTALLATION MANIFESTS IN THE ZTP GITOPS PIPELINE

You can define a set of extra manifests for inclusion in the installation phase of the zero touch provisioning (ZTP) GitOps pipeline. These manifests are linked to the **SiteConfig** custom resources (CRs) and are applied to the cluster during installation. Including **MachineConfig** CRs at install time makes the installation process more efficient.

Prerequisites

- Create a Git repository where you manage your custom site configuration data. The repository must be accessible from the hub cluster and be defined as a source repository for the Argo CD application.

Procedure

1. Create a set of extra manifest CRs that the ZTP pipeline uses to customize the cluster installs.
2. In your custom **/siteconfig** directory, create an **/extra-manifest** folder for your extra manifests. The following example illustrates a sample **/siteconfig** with **/extra-manifest** folder:

```

siteconfig
├── site1-sno-du.yaml
├── site2-standard-du.yaml
├── extra-manifest
│   └── 01-example-machine-config.yaml

```

3. Add your custom extra manifest CRs to the **siteconfig/extra-manifest** directory.
4. In your **SiteConfig** CR, enter the directory name in the **extraManifestPath** field, for example:

```

clusters:
- clusterName: "example-sno"
  networkType: "OVNKubernetes"
  extraManifestPath: extra-manifest

```

5. Save the **SiteConfig** CRs and **/extra-manifest** CRs and push them to the site configuration repo.

The ZTP pipeline appends the CRs in the **/extra-manifest** directory to the default set of extra manifests during cluster provisioning.

23.20. DEPLOYING A SITE

Use the following procedure to prepare the hub cluster for site deployment and initiate zero touch provisioning (ZTP) by pushing custom resources (CRs) to your Git repository.

Procedure

1. Create the required secrets for the site. These resources must be in a namespace with a name matching the cluster name. In **out/argocd/example/siteconfig/example-sno.yaml**, the cluster name and namespace is **example-sno**.

Create the namespace for the cluster using the following commands:

```
$ export CLUSTERSNS=example-sno
```

```
$ oc create namespace $CLUSTERSNS
```

2. Create a pull secret for the cluster. The pull secret must contain all the credentials necessary for installing OpenShift Container Platform and all required Operators. In all of the example **SiteConfig** CRs, the pull secret is named **assisted-deployment-pull-secret**, as shown below:

```

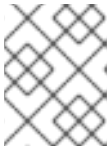
$ oc apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: assisted-deployment-pull-secret
  namespace: $CLUSTERSNS
type: kubernetes.io/dockerconfigjson

```

```
data:
  .dockerconfigjson: $(base64 <pull-secret.json)
EOF
```

3. Create a BMC authentication secret for each host you are deploying:

```
$ oc apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: $(read -p 'Hostname: ' tmp; printf $tmp)-bmc-secret
  namespace: $CLUSTERSNS
type: Opaque
data:
  username: $(read -p 'Username: ' tmp; printf $tmp | base64)
  password: $(read -s -p 'Password: ' tmp; printf $tmp | base64)
EOF
```



NOTE

The secrets are referenced from the **SiteConfig** custom resource (CR) by name. The namespace must match the **SiteConfig** namespace.

4. Create a **SiteConfig** CR for your cluster in your local clone of the Git repository:
 - a. Choose the appropriate example for your CR from the **out/argocd/example/siteconfig/** folder. The folder includes example files for single node, three-node, and standard clusters:
 - **example-sno.yaml**
 - **example-3node.yaml**
 - **example-standard.yaml**
 - b. Change the cluster and host details in the example file to match the type of cluster you want. The following file is a composite of the three files that explains the configuration of each cluster type:

```
# example-node1-bmh-secret & assisted-deployment-pull-secret need to be created
under same namespace example-sno
---
apiVersion: ran.openshift.io/v1
kind: SiteConfig
metadata:
  name: "example-sno"
  namespace: "example-sno"
spec:
  baseDomain: "example.com"
  pullSecretRef:
    name: "assisted-deployment-pull-secret"
  clusterImageSetNameRef: "openshift-4.11" 1
  sshPublicKey: "ssh-rsa AAAA..."
  clusters:
    - clusterName: "example-sno"
      networkType: "OVNKubernetes"
```

```

clusterLabels: 2
  # These example cluster labels correspond to the bindingRules in the
  # PolicyGenTemplate examples in ../policygentemplates:
  # ../policygentemplates/common-ranGen.yaml will apply to all clusters with 'common:
  true'
  common: true
  # ../policygentemplates/group-du-sno-ranGen.yaml will apply to all clusters with
  'group-du-sno: ""'
  group-du-sno: ""
  # ../policygentemplates/example-sno-site.yaml will apply to all clusters with 'sites:
  "example-sno"
  # Normally this should match or contain the cluster name so it only applies to a single
  cluster
  sites : "example-sno"
clusterNetwork:
  - cidr: 1001:1::/48
  hostPrefix: 64
machineNetwork: 3
  - cidr: 1111:2222:3333:4444::/64
  # For 3-node and standard clusters with static IPs, the API and Ingress IPs must be
  configured here
  apiVIP: 1111:2222:3333:4444::1:1 4
  ingressVIP: 1111:2222:3333:4444::1:2 5

serviceNetwork:
  - 1001:2::/112
additionalNTPSources:
  - 1111:2222:3333:4444::2
nodes:
  - hostName: "example-node1.example.com" 6
    role: "master"
    bmcAddress: "idrac-
virtualmedia+https://[1111:2222:3333:4444::bbbb:1]/redfish/v1/Systems/System.Embedded.
1" 7
    bmcCredentialsName:
      name: "example-node1-bmh-secret" 8
    bootMACAddress: "AA:BB:CC:DD:EE:11"
    bootMode: "UEFI"
    rootDeviceHints:
      hctl: '0:1:0'
    cpuset: "0-1,52-53"
    nodeNetwork: 9
      interfaces:
        - name: eno1
          macAddress: "AA:BB:CC:DD:EE:11"
      config:
        interfaces:
          - name: eno1
            type: ethernet
            state: up
            macAddress: "AA:BB:CC:DD:EE:11"
            ipv4:
              enabled: false
            ipv6:
              enabled: true

```

```

address:
  - ip: 1111:2222:3333:4444::1:1
    prefix-length: 64
dns-resolver:
  config:
    search:
      - example.com
    server:
      - 1111:2222:3333:4444::2
routes:
  config:
    - destination: ::0
      next-hop-interface: eno1
      next-hop-address: 1111:2222:3333:4444::1
      table-id: 254

```

- 1 Applies to all cluster types. The value must match an image set available on the hub cluster. To see the list of supported versions on your hub, run **oc get clusterimagesets**.
 - 2 Applies to all cluster types. These values must correspond to the **PolicyGenTemplate** labels that you define in a later step.
 - 3 Applies to single node clusters. The value defines the cluster network sections for a single node deployment.
 - 4 Applies to three-node and standard clusters. The value defines the cluster network sections.
 - 5 Applies to three-node and standard clusters. The value defines the cluster network sections.
 - 6 Applies to all cluster types. For single node deployments, define one host. For three-node deployments, define three hosts. For standard deployments, define three hosts with **role: master** and two or more hosts defined with **role: worker**.
 - 7 Applies to all cluster types. Specifies the BMC address.
 - 8 Applies to all cluster types. Specifies the BMC credentials.
 - 9 Applies to all cluster types. Specifies the network settings for the node.
- c. You can inspect the default set of extra-manifest **MachineConfig** CRs in **out/argocd/extra-manifest**. It is automatically applied to the cluster when it is installed. Optional: To provision additional install-time manifests on the provisioned cluster, create a directory in your Git repository, for example, **sno-extra-manifest/**, and add your custom manifest CRs to this directory. If your **SiteConfig.yaml** refers to this directory in the **extraManifestPath** field, any CRs in this referenced directory are appended to the default set of extra manifests.
5. Add the **SiteConfig** CR to the **kustomization.yaml** file in the **generators** section, similar to the example shown in **out/argocd/example/siteconfig/kustomization.yaml**.
 6. Commit your **SiteConfig** CR and associated **kustomization.yaml** in your Git repository.

7. Push your changes to the Git repository. The ArgoCD pipeline detects the changes and begins the site deployment. You can push the changes to the **SiteConfig** CR and the **PolicyGenTemplate** CR simultaneously.

The **SiteConfig** CR creates the following CRs on the hub cluster:

- **Namespace** - Unique per site
- **AgentClusterInstall**
- **BareMetalHost** - One per node
- **ClusterDeployment**
- **InfraEnv**
- **NMStateConfig** - One per node
- **ExtraManifestsConfigMap** - Extra manifests. The extra manifests include workload partitioning, mountpoint hiding, sctp enablement, and more. To automatically merge the extra manifests into a single manifest per each **MachineConfigPool** role, which is named as **predefined-extra-manifests-`<role>`**, set the **.spec.clusters.mergeDefaultMachineConfigs** to **true** in the **SiteConfig.yaml** file.
- **ManagedCluster**
- **KlusterletAddonConfig**

23.21. GITOPS ZTP AND TOPOLOGY AWARE LIFECYCLE MANAGER

GitOps zero touch provisioning (ZTP) generates installation and configuration CRs from manifests stored in Git. These artifacts are applied to a centralized hub cluster where Red Hat Advanced Cluster Management (RHACM), assisted installer service, and the Topology Aware Lifecycle Manager (TALM) use the CRs to install and configure the spoke cluster. The configuration phase of the ZTP pipeline uses the TALM to orchestrate the application of the configuration CRs to the cluster. There are several key integration points between GitOps ZTP and the TALM.

Inform policies

By default, GitOps ZTP creates all policies with a remediation action of **inform**. These policies cause RHACM to report on compliance status of clusters relevant to the policies but does not apply the desired configuration. During the ZTP installation, the TALM steps through the created **inform** policies, creates a copy for the target spoke cluster(s) and changes the remediation action of the copy to **enforce**. This pushes the configuration to the spoke cluster. Outside of the ZTP phase of the cluster lifecycle, this setup allows changes to be made to policies without the risk of immediately rolling those changes out to all affected spoke clusters in the network. You can control the timing and the set of clusters that are remediated using TALM.

Automatic creation of ClusterGroupUpgrade CRs

The TALM monitors the state of all **ManagedCluster** CRs on the hub cluster. Any **ManagedCluster** CR which does not have a **ztp-done** label applied, including newly created **ManagedCluster** CRs, causes the TALM to automatically create a **ClusterGroupUpgrade** CR with the following characteristics:

- The **ClusterGroupUpgrade** CR is created and enabled in the **ztp-install** namespace.
- **ClusterGroupUpgrade** CR has the same name as the **ManagedCluster** CR.
- The cluster selector includes only the cluster associated with that **ManagedCluster** CR.

- The set of managed policies includes all policies that RHACM has bound to the cluster at the time the **ClusterGroupUpgrade** is created.
- Pre-caching is disabled.
- Timeout set to 4 hours (240 minutes).

The automatic creation of an enabled **ClusterGroupUpgrade** ensures that initial zero-touch deployment of clusters proceeds without the need for user intervention. Additionally, the automatic creation of a **ClusterGroupUpgrade** CR for any **ManagedCluster** without the **ztp-done** label allows a failed ZTP installation to be restarted by simply deleting the **ClusterGroupUpgrade** CR for the cluster.

Waves

Each policy generated from a **PolicyGenTemplate** CR includes a **ztp-deploy-wave** annotation. This annotation is based on the same annotation from each CR which is included in that policy. The wave annotation is used to order the policies in the auto-generated **ClusterGroupUpgrade** CR.



NOTE

All CRs in the same policy must have the same setting for the **ztp-deploy-wave** annotation. The default value of this annotation for each CR can be overridden in the **PolicyGenTemplate**. The wave annotation in the source CR is used for determining and setting the policy wave annotation. This annotation is removed from each built CR which is included in the generated policy at runtime.

The TALM applies the configuration policies in the order specified by the wave annotations. The TALM waits for each policy to be compliant before moving to the next policy. It is important to ensure that the wave annotation for each CR takes into account any prerequisites for those CRs to be applied to the cluster. For example, an Operator must be installed before or concurrently with the configuration for the Operator. Similarly, the **CatalogSource** for an Operator must be installed in a wave before or concurrently with the Operator Subscription. The default wave value for each CR takes these prerequisites into account.

Multiple CRs and policies can share the same wave number. Having fewer policies can result in faster deployments and lower CPU usage. It is a best practice to group many CRs into relatively few waves.

To check the default wave value in each source CR, run the following command against the **out/source-crs** directory that is extracted from the **ztp-site-generate** container image:

```
$ grep -r "ztp-deploy-wave" out/source-crs
```

Phase labels

The **ClusterGroupUpgrade** CR is automatically created and includes directives to annotate the **ManagedCluster** CR with labels at the start and end of the ZTP process.

When ZTP configuration post-installation commences, the **ManagedCluster** has the **ztp-running** label applied. When all policies are remediated to the cluster and are fully compliant, these directives cause the TALM to remove the **ztp-running** label and apply the **ztp-done** label.

For deployments which make use of the **informDuValidator** policy, the **ztp-done** label is applied when the cluster is fully ready for deployment of applications. This includes all reconciliation and resulting effects of the ZTP applied configuration CRs.

Linked CRs

The automatically created **ClusterGroupUpgrade** CR has the owner reference set as the **ManagedCluster** from which it was derived. This reference ensures that deleting the **ManagedCluster** CR causes the instance of the **ClusterGroupUpgrade** to be deleted along with any supporting resources.

23.22. MONITORING DEPLOYMENT PROGRESS

The ArgoCD pipeline uses the **SiteConfig** and **PolicyGenTemplate** CRs in Git to generate the cluster configuration CRs and RHACM policies and then sync them to the hub. You can monitor the progress of this synchronization can be monitored in the ArgoCD dashboard.

Procedure

When the synchronization is complete, the installation generally proceeds as follows:

1. The Assisted Service Operator installs OpenShift Container Platform on the cluster. You can monitor the progress of cluster installation from the RHACM dashboard or from the command line:

```
$ export CLUSTER=<clusterName>
```

```
$ oc get agentclusterinstall -n $CLUSTER $CLUSTER -o jsonpath='{.status.conditions[?(@.type=="Completed")]}' | jq
```

```
$ curl -sk $(oc get agentclusterinstall -n $CLUSTER $CLUSTER -o jsonpath='{.status.debugInfo.eventsURL}') | jq '[-2,-1]'
```

2. The Topology Aware Lifecycle Manager (TALM) applies the configuration policies that are bound to the cluster.

After the cluster installation is complete and the cluster becomes **Ready**, a **ClusterGroupUpgrade** CR corresponding to this cluster, with a list of ordered policies defined by the **ran.openshift.io/ztp-deploy-wave annotations**, is automatically created by the TALM. The cluster's policies are applied in the order listed in **ClusterGroupUpgrade** CR. You can monitor the high-level progress of configuration policy reconciliation using the following commands:

```
$ export CLUSTER=<clusterName>
```

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER -o jsonpath='{.status.conditions[?(@.type=="Ready")]}'
```

3. You can monitor the detailed policy compliant status using the RHACM dashboard or the command line:

```
$ oc get policies -n $CLUSTER
```

The final policy that becomes compliant is the one defined in the ***-du-validator-policy** policies. This policy, when compliant on a cluster, ensures that all cluster configuration, Operator installation, and Operator configuration is complete.

After all policies become complaint, the **ztp-done** label is added to the cluster, indicating the entire ZTP pipeline is complete for the cluster.

23.23. INDICATION OF DONE FOR ZTP INSTALLATIONS

Zero touch provisioning (ZTP) simplifies the process of checking the ZTP installation status for a cluster. The ZTP status moves through three phases: cluster installation, cluster configuration, and ZTP done.

Cluster installation phase

The cluster installation phase is shown by the **ManagedCluster** CR **ManagedClusterJoined** condition. If the **ManagedCluster** CR does not have this condition, or the condition is set to **False**, the cluster is still in the installation phase. Additional details about installation are available from the **AgentClusterInstall** and **ClusterDeployment** CRs. For more information, see "Troubleshooting GitOps ZTP".

Cluster configuration phase

The cluster configuration phase is shown by a **ztp-running** label applied to the **ManagedCluster** CR for the cluster.

ZTP done

Cluster installation and configuration is complete in the ZTP done phase. This is shown by the removal of the **ztp-running** label and addition of the **ztp-done** label to the **ManagedCluster** CR. The **ztp-done** label shows that the configuration has been applied and the baseline DU configuration has completed cluster tuning.

The transition to the ZTP done state is conditional on the compliant state of a Red Hat Advanced Cluster Management (RHACM) static validator inform policy. This policy captures the existing criteria for a completed installation and validates that it moves to a compliant state only when ZTP provisioning of the spoke cluster is complete.

The validator inform policy ensures the configuration of the distributed unit (DU) cluster is fully applied and Operators have completed their initialization. The policy validates the following:

- The target **MachineConfigPool** contains the expected entries and has finished updating. All nodes are available and not degraded.
 - The SR-IOV Operator has completed initialization as indicated by at least one **SriovNetworkNodeState** with **syncStatus: Succeeded**.
 - The PTP Operator daemon set exists.
- The policy captures the existing criteria for a completed installation and validates that it moves to a compliant state only when ZTP provisioning of the spoke cluster is complete.

The validator inform policy is included in the reference group **PolicyGenTemplate** CRs. For reliable indication of the ZTP done state, this validator inform policy must be included in the ZTP pipeline.

23.23.1. Creating a validator inform policy

Use the following procedure to create a validator inform policy that provides an indication of when the zero touch provisioning (ZTP) installation and configuration of the deployed cluster is complete. This policy can be used for deployments of single node clusters, three-node clusters, and standard clusters.

Procedure

1. Create a stand-alone **PolicyGenTemplate** custom resource (CR) that contains the source file **validatorCRs/informDuValidator.yaml**. You only need one stand-alone **PolicyGenTemplate** CR for each cluster type.

Single node clusters

```
group-du-sno-validator-ranGen.yaml
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-du-sno-validator" 1
  namespace: "ztp-group" 2
spec:
  bindingRules:
    group-du-sno: "" 3
  bindingExcludedRules:
    ztp-done: "" 4
  mcp: "master" 5
  sourceFiles:
    - fileName: validatorCRs/informDuValidator.yaml
      remediationAction: inform 6
      policyName: "du-policy" 7
```

Three-node clusters

```
group-du-3node-validator-ranGen.yaml
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-du-3node-validator" 1
  namespace: "ztp-group" 2
spec:
  bindingRules:
    group-du-3node: "" 3
  bindingExcludedRules:
    ztp-done: "" 4
  mcp: "master" 5
  sourceFiles:
    - fileName: validatorCRs/informDuValidator.yaml
      remediationAction: inform 6
      policyName: "du-policy" 7
```

Standard clusters

```
group-du-standard-validator-ranGen.yaml
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "group-du-standard-validator" 1
  namespace: "ztp-group" 2
spec:
  bindingRules:
    group-du-standard: "" 3
  bindingExcludedRules:
    ztp-done: "" 4
  mcp: "worker" 5
```

```
sourceFiles:
- fileName: validatorCRs/informDuValidator.yaml
  remediationAction: inform 6
  policyName: "du-policy" 7
```

- 1 1 1** The name of **PolicyGenTemplates** object. This name is also used as part of the names for the **placementBinding**, **placementRule**, and **policy** that are created in the requested **namespace**.
- 2 2 2** This value should match the **namespace** used in the group **PolicyGenTemplates**.
- 3 3 3** The **group-du-*** label defined in **bindingRules** must exist in the **SiteConfig** files.
- 4 4 4** The label defined in **bindingExcludedRules** must be `ztp-done:`. The **ztp-done** label is used in coordination with the Topology Aware Lifecycle Manager.
- 5 5 5** **mcp** defines the **MachineConfigPool** object that is used in the source file **validatorCRs/informDuValidator.yaml**. It should be **master** for single node and three-node cluster deployments and **worker** for standard cluster deployments.
- 6 6 6** Optional. The default value is **inform**.
- 7 7 7** This value is used as part of the name for the generated RHACM policy. The generated validator policy for the single node example is named **group-du-sno-validator-du-policy**.

2. Push the files to the ZTP Git repository.

23.23.2. Querying the policy compliance status for each cluster

After you have created the validator inform policies for your clusters and pushed them to the zero touch provisioning (ZTP) Git repository, you can check the status of each cluster for policy compliance.

Procedure

1. To query the status of the spoke clusters, use either the Red Hat Advanced Cluster Management (RHACM) web console or the CLI:
 - To query status from the RHACM web console, perform the following actions:
 - a. Click **Governance** → **Find policies**.
 - b. Search for **du-validator-policy**.
 - c. Click into the policy.
 - To query status using the CLI, run the following command:

```
$ oc get policies du-validator-policy -n <namespace_for_common> -o jsonpath=
{'status.status'} | jq
```

When all of the policies including the validator inform policy applied to the cluster become compliant, ZTP installation and configuration for this cluster is complete.

2. To query the cluster violation/compliant status from the ACM web console, click **Governance** → **Cluster violations**.
3. Check the validator policy compliant status for a cluster using the following commands:
 - a. Export the cluster name:

```
$ export CLUSTER=<cluster_name>
```

- b. Get the policy:

```
$ oc get policies -n $CLUSTER | grep <validator_policy_name>
```

Alternatively, you can use the following command:

```
$ oc get policies -n <namespace-for-group> <validatorPolicyName> -o jsonpath="{.status.status[?(@.clustername=='$CLUSTER')]}" | jq
```

After the ***-validator-du-policy** RHACM policy becomes compliant for the cluster, the validator policy is unbound for this cluster and the **ztp-done** label is added to the cluster. This acts as a persistent indicator that the whole ZTP pipeline has completed for the cluster.

23.23.3. Node Tuning Operator

The Node Tuning Operator provides the ability to enable advanced node performance tunings on a set of nodes.

OpenShift Container Platform provides a 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 updating the kernel to **rt-kernel**, reserving CPUs for management workloads, and using CPUs for running the workloads.



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, these functions are part of the Node Tuning Operator.

23.24. TROUBLESHOOTING GITOPS ZTP

The ArgoCD pipeline uses the **SiteConfig** and **PolicyGenTemplate** custom resources (CRs) from Git to generate the cluster configuration CRs and Red Hat Advanced Cluster Management (RHACM) policies. Use the following steps to troubleshoot issues that might occur during this process.

file// Module included in the following assemblies:

23.24.1. Validating the generation of installation CRs

The GitOps zero touch provisioning (ZTP) infrastructure generates a set of installation CRs on the hub cluster in response to a **SiteConfig** CR pushed to your Git repository. You can check that the installation CRs were created by using the following command:

```
$ oc get AgentClusterInstall -n <cluster_name>
```

If no object is returned, use the following procedure to troubleshoot the ArgoCD pipeline flow from **SiteConfig** files to the installation CRs.

Procedure

1. Verify that the **SiteConfig** → **ManagedCluster** was generated to the hub cluster:

```
$ oc get managedcluster
```

2. If the **SiteConfig ManagedCluster** is missing, see if the **clusters** application failed to synchronize the files from the Git repository to the hub:

```
$ oc describe -n openshift-gitops application clusters
```

3. Check for **Status: Conditions:** to view the error logs. For example, setting an invalid value for **extraManifestPath:** in the **siteConfig** file raises an error as shown below:

```
Status:
Conditions:
  Last Transition Time: 2021-11-26T17:21:39Z
  Message:           rpc error: code = Unknown desc = `kustomize build
/tmp/https___git.com/ran-sites/siteconfigs/ --enable-alpha-plugins` failed exit status 1:
2021/11/26 17:21:40 Error could not create extra-manifest ranSite1.extra-manifest3 stat
extra-manifest3: no such file or directory
2021/11/26 17:21:40 Error: could not build the entire SiteConfig defined by /tmp/kust-plugin-
config-913473579: stat extra-manifest3: no such file or directory
Error: failure in plugin configured via /tmp/kust-plugin-config-913473579; exit status 1: exit
status 1
Type: ComparisonError
```

4. Check for **Status: Sync:**. If there are log errors, **Status: Sync:** could indicate an **Unknown** error:

```
Status:
Sync:
  Compared To:
    Destination:
      Namespace: clusters-sub
      Server:    https://kubernetes.default.svc
    Source:
      Path:      sites-config
      Repo URL:  https://git.com/ran-sites/siteconfigs/.git
      Target Revision: master
  Status:      Unknown
```

23.24.2. Validating the generation of configuration policy CRs

Policy custom resources (CRs) are generated in the same namespace as the **PolicyGenTemplate** from which they are created. The same troubleshooting flow applies to all policy CRs generated from a **PolicyGenTemplate** regardless of whether they are **ztp-common**, **ztp-group**, or **ztp-site** based, as shown using the following commands:

```
$ export NS=<namespace>
```

```
$ oc get policy -n $NS
```

The expected set of policy-wrapped CRs should be displayed.

If the policies failed synchronization, use the following troubleshooting steps.

Procedure

1. To display detailed information about the policies, run the following command:

```
$ oc describe -n openshift-gitops application policies
```

2. Check for **Status: Conditions:** to show the error logs. For example, setting an invalid **sourceFile** → **fileName:** generates the error shown below:

```
Status:
Conditions:
  Last Transition Time: 2021-11-26T17:21:39Z
  Message:           rpc error: code = Unknown desc = `kustomize build
/tmp/https___git.com/ran-sites/policies/ --enable-alpha-plugins` failed exit status 1:
2021/11/26 17:21:40 Error could not find test.yaml under source-crs/: no such file or directory
Error: failure in plugin configured via /tmp/kust-plugin-config-52463179; exit status 1: exit
status 1
  Type: ComparisonError
```

3. Check for **Status: Sync:**. If there are log errors at **Status: Conditions:**, the **Status: Sync:** shows **Unknown** or **Error**:

```
Status:
Sync:
  Compared To:
  Destination:
    Namespace: policies-sub
    Server:    https://kubernetes.default.svc
  Source:
    Path:      policies
    Repo URL:  https://git.com/ran-sites/policies/.git
    Target Revision: master
  Status:      Error
```

4. When Red Hat Advanced Cluster Management (RHACM) recognizes that policies apply to a **ManagedCluster** object, the policy CR objects are applied to the cluster namespace. Check to see if the policies were copied to the cluster namespace:

```
$ oc get policy -n $CLUSTER
```


Example output:

NAME	REMEDIATION ACTION	COMPLIANCE STATE	AGE
ztp-common.common-config-policy	inform	Compliant	13d
ztp-common.common-subscriptions-policy	inform	Compliant	13d
ztp-group.group-du-sno-config-policy	inform	Compliant	13d
Ztp-group.group-du-sno-validator-du-policy	inform	Compliant	13d
ztp-site.example-sno-config-policy	inform	Compliant	13d

RHACM copies all applicable policies into the cluster namespace. The copied policy names have the format: **<policyGenTemplate.Namespace>.<policyGenTemplate.Name>-<policyName>**.

5. Check the placement rule for any policies not copied to the cluster namespace. The **matchSelector** in the **PlacementRule** for those policies should match labels on the **ManagedCluster** object:

```
$ oc get placementrule -n $NS
```

6. Note the **PlacementRule** name appropriate for the missing policy, common, group, or site, using the following command:

```
$ oc get placementrule -n $NS <placementRuleName> -o yaml
```

- The status-decisions should include your cluster name.
- The key-value pair of the **matchSelector** in the spec must match the labels on your managed cluster.

7. Check the labels on the **ManagedCluster** object using the following command:

```
$ oc get ManagedCluster $CLUSTER -o jsonpath='{.metadata.labels}' | jq
```

8. Check to see which policies are compliant using the following command:

```
$ oc get policy -n $CLUSTER
```

If the **Namespace**, **OperatorGroup**, and **Subscription** policies are compliant but the Operator configuration policies are not, it is likely that the Operators did not install on the spoke cluster. This causes the Operator configuration policies to fail to apply because the CRD is not yet applied to the spoke.

23.24.3. Restarting policies reconciliation

Use the following procedure to restart policies reconciliation in the event of unexpected compliance issues. This procedure is required when the **ClusterGroupUpgrade** CR has timed out.

Procedure

1. A **ClusterGroupUpgrade** CR is generated in the namespace **ztp-install** by the Topology Aware Lifecycle Manager after the managed spoke cluster becomes **Ready**:

```
$ export CLUSTER=<clusterName>
```

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER
```

2. If there are unexpected issues and the policies fail to become compliant within the configured timeout (the default is 4 hours), the status of the **ClusterGroupUpgrade** CR shows **UpgradeTimedOut**:

```
$ oc get clustergroupupgrades -n ztp-install $CLUSTER -o jsonpath='{.status.conditions[?(@.type=="Ready")]}'
```

3. A **ClusterGroupUpgrade** CR in the **UpgradeTimedOut** state automatically restarts its policy reconciliation every hour. If you have changed your policies, you can start a retry immediately by deleting the existing **ClusterGroupUpgrade** CR. This triggers the automatic creation of a new **ClusterGroupUpgrade** CR that begins reconciling the policies immediately:

```
$ oc delete clustergroupupgrades -n ztp-install $CLUSTER
```

Note that when the **ClusterGroupUpgrade** CR completes with status **UpgradeCompleted** and the managed spoke cluster has the label **ztp-done** applied, you can make additional configuration changes using **PolicyGenTemplate**. Deleting the existing **ClusterGroupUpgrade** CR will not make the TALM generate a new CR.

At this point, ZTP has completed its interaction with the cluster and any further interactions should be treated as an upgrade.

Additional resources

- For information about using TALM to construct your own **ClusterGroupUpgrade** CR, see [About the ClusterGroupUpgrade CR](#).

23.25. SITE CLEANUP

Remove a site and the associated installation and configuration policy CRs by removing the **SiteConfig** and **PolicyGenTemplate** file names from the **kustomization.yaml** file. When you run the ZTP pipeline again, the generated CRs are removed. If you want to permanently remove a site, you should also remove the **SiteConfig** and site-specific **PolicyGenTemplate** files from the Git repository. If you want to remove a site temporarily, for example when redeploying a site, you can leave the **SiteConfig** and site-specific **PolicyGenTemplate** CRs in the Git repository.



NOTE

After removing the **SiteConfig** file, if the corresponding clusters remain in the detach process, check Red Hat Advanced Cluster Management (RHACM) for information about cleaning up the detached managed cluster.

Additional resources

- For information about removing a cluster, see [Removing a cluster from management](#).

23.25.1. Removing obsolete content

If a change to the **PolicyGenTemplate** file configuration results in obsolete policies, for example, policies are renamed, use the following procedure to remove those policies in an automated way.

Procedure

1. Remove the affected **PolicyGenTemplate** files from the Git repository, commit and push to the remote repository.
2. Wait for the changes to synchronize through the application and the affected policies to be removed from the hub cluster.
3. Add the updated **PolicyGenTemplate** files back to the Git repository, and then commit and push to the remote repository.

Note that removing the zero touch provisioning (ZTP) distributed unit (DU) profile policies from the Git repository, and as a result also removing them from the hub cluster, does not affect any configuration of the managed spoke clusters. Removing a policy from the hub cluster does not delete it from the spoke cluster and the CRs managed by that policy.

As an alternative, after making changes to **PolicyGenTemplate** files that result in obsolete policies, you can remove these policies from the hub cluster manually. You can delete policies from the RHACM console using the **Governance** tab or by using the following command:

```
$ oc delete policy -n <namespace> <policyName>
```

23.25.2. Tearing down the pipeline

If you need to remove the ArgoCD pipeline and all generated artifacts follow this procedure:

Procedure

1. Detach all clusters from RHACM.
2. Delete the **kustomization.yaml** file in the **deployment** directory using the following command:

```
$ oc delete -k out/argocd/deployment
```

23.26. UPGRADING GITOPS ZTP

You can upgrade the Gitops zero touch provisioning (ZTP) infrastructure independently from the underlying cluster, Red Hat Advanced Cluster Management (RHACM), and OpenShift Container Platform version running on the spoke clusters. This procedure guides you through the upgrade process to avoid impact on the spoke clusters. However, any changes to the content or settings of policies, including adding recommended content, results in changes that must be rolled out and reconciled to the spoke clusters.

Prerequisites

- This procedure assumes that you have a fully operational hub cluster running the earlier version of the GitOps ZTP infrastructure.

Procedure

At a high level, the strategy for upgrading the GitOps ZTP infrastructure is:

1. Label all existing clusters with the **ztp-done** label.
2. Stop the ArgoCD applications.

3. Install the new tooling.
4. Update required content and optional changes in the Git repository.
5. Update and restart the application configuration.

23.26.1. Preparing for the upgrade

Use the following procedure to prepare your site for the GitOps zero touch provisioning (ZTP) upgrade.

Procedure

1. Obtain the latest version of the GitOps ZTP container from which you can extract a set of custom resources (CRs) used to configure the GitOps operator on the hub cluster for use in the GitOps ZTP solution.
2. Extract the **argocd/deployment** directory using the following commands:

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

```
$ podman run --log-driver=none --rm registry.redhat.io/openshift4/ztp-site-generate-rhel8:v4.11 extract /home/ztp --tar | tar x -C ./out
```

The **/out** directory contains the following subdirectories:

- **out/extra-manifest**: contains the source CR files that the **SiteConfig** CR uses to generate the extra manifest **configMap**.
 - **out/source-crs**: contains the source CR files that the **PolicyGenTemplate** CR uses to generate the Red Hat Advanced Cluster Management (RHACM) policies.
 - **out/argocd/deployment**: contains patches and YAML files to apply on the hub cluster for use in the next step of this procedure.
 - **out/argocd/example**: contains example **SiteConfig** and **PolicyGenTemplate** files that represent the recommended configuration.
3. Update the **clusters-app.yaml** and **policies-app.yaml** files to reflect the name of your applications and the URL, branch, and path for your Git repository.

If the upgrade includes changes to policies that may result in obsolete policies, these policies should be removed prior to performing the upgrade.

23.26.2. Labeling the existing clusters

To ensure that existing clusters remain untouched by the tooling updates, all existing managed clusters must be labeled with the **ztp-done** label.

Procedure

1. Find a label selector that lists the managed clusters that were deployed with zero touch provisioning (ZTP), such as **local-cluster!=true**:

```
$ oc get managedcluster -l 'local-cluster!=true'
```

2. Ensure that the resulting list contains all the managed clusters that were deployed with ZTP, and then use that selector to add the **ztp-done** label:

```
$ oc label managedcluster -l 'local-cluster!=true' ztp-done=
```

23.26.3. Stopping the existing GitOps ZTP applications

Removing the existing applications ensures that any changes to existing content in the Git repository are not rolled out until the new version of the tooling is available.

Use the application files from the **deployment** directory. If you used custom names for the applications, update the names in these files first.

Procedure

1. Perform a non-cascaded delete on the **clusters** application to leave all generated resources in place:

```
$ oc delete -f out/argocd/deployment/clusters-app.yaml
```

2. Perform a cascaded delete on the **policies** application to remove all previous policies:

```
$ oc patch -f policies-app.yaml -p '{"metadata": {"finalizers": ["resources-finalizer.argocd.argoproj.io"]}}' --type merge
```

```
$ oc delete -f out/argocd/deployment/policies-app.yaml
```

23.26.4. Topology Aware Lifecycle Manager

Install the Topology Aware Lifecycle Manager (TALM) on the hub cluster.

Additional resources

- For information about the Topology Aware Lifecycle Manager (TALM), see [About the Topology Aware Lifecycle Manager configuration](#).

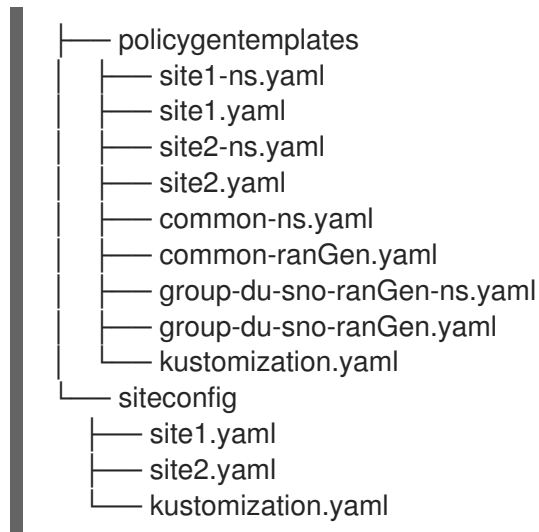
23.26.5. Required changes to the Git repository

When upgrading the **ztp-site-generate** container from an earlier release to the 4.10 version, additional requirements are placed on the contents of the Git repository. Existing content in the repository must be updated to reflect these changes.

- Changes to **PolicyGenTemplate** files:
All **PolicyGenTemplate** files must be created in a **Namespace** prefixed with **ztp**. This ensures that the GitOps zero touch provisioning (ZTP) application is able to manage the policy CRs generated by GitOps ZTP without conflicting with the way Red Hat Advanced Cluster Management (RHACM) manages the policies internally.
- Remove the **pre-sync.yaml** and **post-sync.yaml** files:
This step is optional but recommended. When the **kustomization.yaml** files are added, the **pre-sync.yaml** and **post-sync.yaml** files are no longer used. They must be removed to avoid confusion and can potentially cause errors if kustomization files are inadvertently removed.

Note that there is a set of **pre-sync.yaml** and **post-sync.yaml** files under both the **SiteConfig** and **PolicyGenTemplate** trees.

- Add the **kustomization.yaml** file to the repository:
All **SiteConfig** and **PolicyGenTemplate** CRs must be included in a **kustomization.yaml** file under their respective directory trees. For example:



NOTE

The files listed in the **generator** sections must contain either **SiteConfig** or **PolicyGenTemplate** CRs only. If your existing YAML files contain other CRs, for example, **Namespace**, these other CRs must be pulled out into separate files and listed in the **resources** section.

The **PolicyGenTemplate** kustomization file must contain all **PolicyGenTemplate** YAML files in the **generator** section and **Namespace** CRs in the **resources** section. For example:

```

apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

generators:
- common-ranGen.yaml
- group-du-sno-ranGen.yaml
- site1.yaml
- site2.yaml

resources:
- common-ns.yaml
- group-du-sno-ranGen-ns.yaml
- site1-ns.yaml
- site2-ns.yaml
  
```

The **SiteConfig** kustomization file must contain all **SiteConfig** YAML files in the **generator** section and any other CRs in the resources:

```

apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
  
```

```
generators:
```

- site1.yaml
- site2.yaml

- Review and incorporate recommended changes

Each release may include additional recommended changes to the configuration applied to deployed clusters. Typically these changes result in lower CPU use by the OpenShift platform, additional features, or improved tuning of the platform.

Review the reference **SiteConfig** and **PolicyGenTemplate** CRs applicable to the types of cluster in your network. These examples can be found in the **argocd/example** directory extracted from the GitOps ZTP container.

23.26.6. Installing the new GitOps ZTP applications

Using the extracted **argocd/deployment** directory, and after ensuring that the applications point to your Git repository, apply the full contents of the deployment directory. Applying the full contents of the directory ensures that all necessary resources for the applications are correctly configured.

Procedure

1. To patch the ArgoCD instance in the hub cluster by using the patch file previously extracted into the **out/argocd/deployment/** directory, enter the following command:

```
$ oc patch argocd openshift-gitops \
-n openshift-gitops --type=merge \
--patch-file out/argocd/deployment/argocd-openshift-gitops-patch.json
```

2. To apply the contents of the **argocd/deployment** directory, enter the following command:

```
$ oc apply -k out/argocd/deployment
```

23.26.7. Roll out the configuration changes

If any configuration changes were included in the upgrade due to implementing recommended changes, the upgrade process results in a set of policy CRs on the hub cluster in the **Non-Compliant** state. As of the OpenShift Container Platform 4.10 version of the **ztp-site-generate** container, these policies are set to **inform** mode and are not pushed to the spoke clusters without an additional step by the user. This ensures that potentially disruptive changes to the clusters can be managed in terms of when the changes are made, for example, during a maintenance window, and how many clusters are updated concurrently.

To roll out the changes, create one or more **ClusterGroupUpgrade** CRs as detailed in the TALM documentation. The CR must contain the list of **Non-Compliant** policies that you want to push out to the spoke clusters as well as a list or selector of which clusters should be included in the update.

Additional resources

- For information about creating **ClusterGroupUpgrade** CRs, see [About the auto-created ClusterGroupUpgrade CR for ZTP](#).

23.27. MANUALLY INSTALL A SINGLE MANAGED CLUSTER

This procedure tells you how to manually create and deploy a single managed cluster. If you are creating multiple clusters, perhaps hundreds, use the **SiteConfig** method described in “Creating ZTP custom resources for multiple managed clusters”.

Prerequisites

- Enable the Assisted Installer service.
- Ensure network connectivity:
 - The container within the hub must be able to reach the Baseboard Management Controller (BMC) address of the target bare-metal host.
 - The managed cluster must be able to resolve and reach the hub’s API **hostname** and ***.app** hostname. Here is an example of the hub’s API and ***.app** hostname:

```
console-openshift-console.apps.hub-cluster.internal.domain.com
api.hub-cluster.internal.domain.com
```

- The hub must be able to resolve and reach the API and ***.app** hostname of the managed cluster. Here is an example of the managed cluster’s API and ***.app** hostname:

```
console-openshift-console.apps.sno-managed-cluster-1.internal.domain.com
api.sno-managed-cluster-1.internal.domain.com
```

- A DNS server that is IP reachable from the target bare-metal host.
- A target bare-metal host for the managed cluster with the following hardware minimums:
 - 4 CPU or 8 vCPU
 - 32 GiB RAM
 - 120 GiB disk for root file system
- When working in a disconnected environment, the release image must be mirrored. Use this command to mirror the release image:

```
$ oc adm release mirror -a <pull_secret.json>
--from=quay.io/openshift-release-dev/ocp-release:{{ mirror_version_spoke_release }}
--to={{ provisioner_cluster_registry }}/ocp4 --to-release-image={{
provisioner_cluster_registry }}/ocp4:{{ mirror_version_spoke_release }}
```

- You mirrored the ISO and **rootfs** used to generate the spoke cluster ISO to an HTTP server and configured the settings to pull images from there.
The images must match the version of the **ClusterImageSet**. For example, to deploy a 4.11.0 version, the **rootfs** and ISO must be set to **4.11.0**.

Procedure

1. Create a **ClusterImageSet** for each specific cluster version that needs to be deployed. A **ClusterImageSet** has the following format:

```
apiVersion: hive.openshift.io/v1
kind: ClusterImageSet
```



```

metadata:
  name: openshift-4.11.0 1
spec:
  releaseImage: quay.io/openshift-release-dev/ocp-release:4.11.0-x86_64 2

```

- 1** The descriptive version that you want to deploy.
- 2** Specifies the **releaseImage** to deploy and determines the OS Image version. The discovery ISO is based on an OS image version as the **releaseImage**, or latest if the exact version is unavailable.

2. Create the **Namespace** definition for the managed cluster:

```

apiVersion: v1
kind: Namespace
metadata:
  name: <cluster_name> 1
  labels:
    name: <cluster_name> 2

```

- 1** **2** The name of the managed cluster to provision.

3. Create the **BMC Secret** custom resource:

```

apiVersion: v1
data:
  password: <bmc_password> 1
  username: <bmc_username> 2
kind: Secret
metadata:
  name: <cluster_name>-bmc-secret
  namespace: <cluster_name>
type: Opaque

```

- 1** The password to the target bare-metal host. Must be base-64 encoded.
- 2** The username to the target bare-metal host. Must be base-64 encoded.

4. Create the **Image Pull Secret** custom resource:

```

apiVersion: v1
data:
  .dockerconfigjson: <pull_secret> 1
kind: Secret
metadata:
  name: assisted-deployment-pull-secret
  namespace: <cluster_name>
type: kubernetes.io/dockerconfigjson

```

- 1** The OpenShift Container Platform pull secret. Must be base-64 encoded.

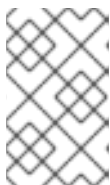
5. Create the **AgentClusterInstall** custom resource:

```

apiVersion: extensions.hive.openshift.io/v1beta1
kind: AgentClusterInstall
metadata:
  # Only include the annotation if using OVN, otherwise omit the annotation
  annotations:
    agent-install.openshift.io/install-config-overrides: '{"networking":
{"networkType":"OVNKubernetes"}}'
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  clusterDeploymentRef:
    name: <cluster_name>
  imageSetRef:
    name: <cluster_image_set> ❶
  networking:
    clusterNetwork:
      - cidr: <cluster_network_cidr> ❷
        hostPrefix: 23
    machineNetwork:
      - cidr: <machine_network_cidr> ❸
    serviceNetwork:
      - <service_network_cidr> ❹
  provisionRequirements:
    controlPlaneAgents: 1
    workerAgents: 0
  sshPublicKey: <public_key> ❺

```

- ❶ The name of the **ClusterImageSet** custom resource used to install OpenShift Container Platform on the bare-metal host.
- ❷ A block of IPv4 or IPv6 addresses in CIDR notation used for communication among cluster nodes.
- ❸ A block of IPv4 or IPv6 addresses in CIDR notation used for the target bare-metal host external communication. Also used to determine the API and Ingress VIP addresses when provisioning DU single-node clusters.
- ❹ A block of IPv4 or IPv6 addresses in CIDR notation used for cluster services internal communication.
- ❺ A plain text string. You can use the public key to SSH into the node after it has finished installing.

**NOTE**

If you want to configure a static IP address for the managed cluster at this point, see the procedure in this document for configuring static IP addresses for managed clusters.

6. Create the **ClusterDeployment** custom resource:

```

apiVersion: hive.openshift.io/v1

```

```

kind: ClusterDeployment
metadata:
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  baseDomain: <base_domain> ❶
  clusterInstallRef:
    group: extensions.hive.openshift.io
    kind: AgentClusterInstall
    name: <cluster_name>
    version: v1beta1
  clusterName: <cluster_name>
  platform:
    agentBareMetal:
      agentSelector:
        matchLabels:
          cluster-name: <cluster_name>
  pullSecretRef:
    name: assisted-deployment-pull-secret

```

❶ The managed cluster's base domain.

7. Create the **KlusterletAddonConfig** custom resource:

```

apiVersion: agent.open-cluster-management.io/v1
kind: KlusterletAddonConfig
metadata:
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  clusterName: <cluster_name>
  clusterNamespace: <cluster_name>
  clusterLabels:
    cloud: auto-detect
    vendor: auto-detect
  applicationManager:
    enabled: true
  certPolicyController:
    enabled: false
  iamPolicyController:
    enabled: false
  policyController:
    enabled: true
  searchCollector:
    enabled: false ❶

```

❶ Keep **searchCollector** disabled. Set to **true** to enable the **KlusterletAddonConfig** CR or **false** to disable the **KlusterletAddonConfig** CR.

8. Create the **ManagedCluster** custom resource:

```

apiVersion: cluster.open-cluster-management.io/v1
kind: ManagedCluster
metadata:

```

```

name: <cluster_name>
spec:
  hubAcceptsClient: true

```

9. Create the **InfraEnv** custom resource:

```

apiVersion: agent-install.openshift.io/v1beta1
kind: InfraEnv
metadata:
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  clusterRef:
    name: <cluster_name>
    namespace: <cluster_name>
  sshAuthorizedKey: <public_key> ❶
  agentLabelSelector:
    matchLabels:
      cluster-name: <cluster_name>
  pullSecretRef:
    name: assisted-deployment-pull-secret

```

- ❶ Entered as plain text. You can use the public key to SSH into the target bare-metal host when it boots from the ISO.

10. Create the **BareMetalHost** custom resource:

```

apiVersion: metal3.io/v1alpha1
kind: BareMetalHost
metadata:
  name: <cluster_name>
  namespace: <cluster_name>
  annotations:
    inspect.metal3.io: disabled
  labels:
    infraenvs.agent-install.openshift.io: "<cluster_name>"
spec:
  bootMode: "UEFI"
  bmc:
    address: <bmc_address> ❶
    disableCertificateVerification: true
    credentialsName: <cluster_name>-bmc-secret
  bootMACAddress: <mac_address> ❷
  automatedCleaningMode: disabled
  online: true

```

- ❶ The baseboard management console address of the installation ISO on the target bare-metal host.

- ❷ The MAC address of the target bare-metal host.

Optionally, you can add **bmac.agent-install.openshift.io/hostname: <host-name>** as an annotation to set the managed cluster's hostname. If you don't add the annotation, the hostname will default to either a hostname from the DHCP server or local host.

11. After you have created the custom resources, push the entire directory of generated custom resources to the Git repository you created for storing the custom resources.

Next steps

To provision additional clusters, repeat this procedure for each cluster.

23.27.1. Configuring BIOS for distributed unit bare-metal hosts

Distributed unit (DU) hosts require the BIOS to be configured before the host can be provisioned. The BIOS configuration is dependent on the specific hardware that runs your DUs and the particular requirements of your installation.

Procedure

1. Set the **UEFI/BIOS Boot Mode** to **UEFI**.
2. In the host boot sequence order, set **Hard drive first**.
3. Apply the specific BIOS configuration for your hardware. The following table describes a representative BIOS configuration for an Intel Xeon Skylake or Intel Cascade Lake server, based on the Intel FlexRAN 4G and 5G baseband PHY reference design.



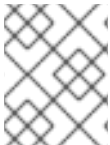
IMPORTANT

The exact BIOS configuration depends on your specific hardware and network requirements. The following sample configuration is for illustrative purposes only.

Table 23.2. Sample BIOS configuration for an Intel Xeon Skylake or Cascade Lake server

BIOS Setting	Configuration
CPU Power and Performance Policy	Performance
Uncore Frequency Scaling	Disabled
Performance P-limit	Disabled
Enhanced Intel SpeedStep® Tech	Enabled
Intel Configurable TDP	Enabled
Configurable TDP Level	Level 2
Intel® Turbo Boost Technology	Enabled
Energy Efficient Turbo	Disabled

BIOS Setting	Configuration
Hardware P-States	Disabled
Package C-State	C0/C1 state
C1E	Disabled
Processor C6	Disabled

**NOTE**

Enable global SR-IOV and VT-d settings in the BIOS for the host. These settings are relevant to bare-metal environments.

23.27.2. Configuring static IP addresses for managed clusters

Optionally, after creating the **AgentClusterInstall** custom resource, you can configure static IP addresses for the managed clusters.

**NOTE**

You must create this custom resource before creating the **ClusterDeployment** custom resource.

Prerequisites

- Deploy and configure the **AgentClusterInstall** custom resource.

Procedure

1. Create a **NMStateConfig** custom resource:

```
apiVersion: agent-install.openshift.io/v1beta1
kind: NMStateConfig
metadata:
  name: <cluster_name>
  namespace: <cluster_name>
  labels:
    sno-cluster-<cluster-name>: <cluster_name>
spec:
  config:
    interfaces:
      - name: eth0
        type: ethernet
        state: up
        ipv4:
          enabled: true
          address:
            - ip: <ip_address> 1
              prefix-length: <public_network_prefix> 2
```

```

    dhcp: false
  dns-resolver:
    config:
      server:
        - <dns_resolver> ❸
  routes:
    config:
      - destination: 0.0.0.0/0
        next-hop-address: <gateway> ❹
        next-hop-interface: eth0
        table-id: 254
  interfaces:
    - name: "eth0" ❺
      macAddress: <mac_address> ❻

```

- ❶ The static IP address of the target bare-metal host.
 - ❷ The static IP address's subnet prefix for the target bare-metal host.
 - ❸ The DNS server for the target bare-metal host.
 - ❹ The gateway for the target bare-metal host.
 - ❺ Must match the name specified in the **interfaces** section.
 - ❻ The mac address of the interface.
2. When creating the **BareMetalHost** custom resource, ensure that one of its mac addresses matches a mac address in the **NMStateConfig** target bare-metal host.
 3. When creating the **InfraEnv** custom resource, reference the label from the **NMStateConfig** custom resource in the **InfraEnv** custom resource:

```

apiVersion: agent-install.openshift.io/v1beta1
kind: InfraEnv
metadata:
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  clusterRef:
    name: <cluster_name>
    namespace: <cluster_name>
  sshAuthorizedKey: <public_key>
  agentLabelSelector:
    matchLabels:
      cluster-name: <cluster_name>
  pullSecretRef:
    name: assisted-deployment-pull-secret
  nmStateConfigLabelSelector:
    matchLabels:
      sno-cluster-<cluster-name>: <cluster_name> # Match this label

```

23.27.3. Automated Discovery image ISO process for provisioning clusters

After you create the custom resources, the following actions happen automatically:

1. A Discovery image ISO file is generated and booted on the target machine.
2. When the ISO file successfully boots on the target machine it reports the hardware information of the target machine.
3. After all hosts are discovered, OpenShift Container Platform is installed.
4. When OpenShift Container Platform finishes installing, the hub installs the **klusterlet** service on the target cluster.
5. The requested add-on services are installed on the target cluster.

The Discovery image ISO process finishes when the **Agent** custom resource is created on the hub for the managed cluster.

23.27.4. Checking the managed cluster status

Ensure that cluster provisioning was successful by checking the cluster status.

Prerequisites

- All of the custom resources have been configured and provisioned, and the **Agent** custom resource is created on the hub for the managed cluster.

Procedure

1. Check the status of the managed cluster:

```
$ oc get managedcluster
```

True indicates the managed cluster is ready.

2. Check the agent status:

```
$ oc get agent -n <cluster_name>
```

3. Use the **describe** command to provide an in-depth description of the agent's condition. Statuses to be aware of include **BackendError**, **InputError**, **ValidationsFailing**, **InstallationFailed**, and **AgentIsConnected**. These statuses are relevant to the **Agent** and **AgentClusterInstall** custom resources.

```
$ oc describe agent -n <cluster_name>
```

4. Check the cluster provisioning status:

```
$ oc get agentclusterinstall -n <cluster_name>
```

5. Use the **describe** command to provide an in-depth description of the cluster provisioning status:

```
$ oc describe agentclusterinstall -n <cluster_name>
```


- Check the status of the managed cluster's add-on services:

```
$ oc get managedclusteraddon -n <cluster_name>
```

- Retrieve the authentication information of the **kubeconfig** file for the managed cluster:

```
$ oc get secret -n <cluster_name> <cluster_name>-admin-kubeconfig -o jsonpath={.data.kubeconfig} | base64 -d > <directory>/<cluster_name>-kubeconfig
```

23.27.5. Configuring a managed cluster for a disconnected environment

After you have completed the preceding procedure, follow these steps to configure the managed cluster for a disconnected environment.

Prerequisites

- A disconnected installation of Red Hat Advanced Cluster Management (RHACM) 2.3.
- Host the **rootfs** and **iso** images on an HTTPD server.



WARNING

If you enable TLS for the HTTPD server, you must confirm the root certificate is signed by an authority trusted by the client and verify the trusted certificate chain between your OpenShift Container Platform hub and spoke clusters and the HTTPD server. Using a server configured with an untrusted certificate prevents the images from being downloaded to the image creation service. Using untrusted HTTPS servers is not supported.

Procedure

- Create a **ConfigMap** containing the mirror registry config:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: assisted-installer-mirror-config
  namespace: assisted-installer
  labels:
    app: assisted-service
data:
  ca-bundle.crt: <certificate> ❶
  registries.conf: | ❷
    unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

  [[registry]]
    location = <mirror_registry_url> ❸
    insecure = false
    mirror-by-digest-only = true
```

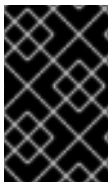
- 1 The mirror registry's certificate used when creating the mirror registry.
- 2 The configuration for the mirror registry.
- 3 The URL of the mirror registry.

This updates **mirrorRegistryRef** in the **AgentServiceConfig** custom resource, as shown below:

Example output

```
apiVersion: agent-install.openshift.io/v1beta1
kind: AgentServiceConfig
metadata:
  name: agent
  namespace: assisted-installer
spec:
  databaseStorage:
    volumeName: <db_pv_name>
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <db_storage_size>
  filesystemStorage:
    volumeName: <fs_pv_name>
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <fs_storage_size>
  mirrorRegistryRef:
    name: 'assisted-installer-mirror-config'
  osImages:
    - openshiftVersion: <ocp_version>
      rootfs: <rootfs_url> 1
      url: <iso_url> 2
```

- 1 2 Must match the URLs of the HTTPD server.



IMPORTANT

A valid NTP server is required during cluster installation. Ensure that a suitable NTP server is available and can be reached from the installed clusters through the disconnected network.

23.27.6. Configuring IPv6 addresses for a disconnected environment

Optionally, when you are creating the **AgentClusterInstall** custom resource, you can configure IPv6 addresses for the managed clusters.

Procedure

1. In the **AgentClusterInstall** custom resource, modify the IP addresses in **clusterNetwork** and **serviceNetwork** for IPv6 addresses:

```
apiVersion: extensions.hive.openshift.io/v1beta1
kind: AgentClusterInstall
metadata:
  # Only include the annotation if using OVN, otherwise omit the annotation
  annotations:
    agent-install.openshift.io/install-config-overrides: '{"networking":
{"networkType":"OVNKubernetes"}}'
  name: <cluster_name>
  namespace: <cluster_name>
spec:
  clusterDeploymentRef:
    name: <cluster_name>
  imageSetRef:
    name: <cluster_image_set>
  networking:
    clusterNetwork:
      - cidr: "fd01::/48"
        hostPrefix: 64
    machineNetwork:
      - cidr: <machine_network_cidr>
    serviceNetwork:
      - "fd02::/112"
  provisionRequirements:
    controlPlaneAgents: 1
    workerAgents: 0
    sshPublicKey: <public_key>
```

2. Update the **NMStateConfig** custom resource with the IPv6 addresses you defined.

23.28. GENERATING RAN POLICIES

Prerequisites

- Install Kustomize
- Install the [Kustomize Policy Generator plug-in](#)

Procedure

1. Configure the **kustomization.yaml** file to reference the **policyGenerator.yaml** file. The following example shows the PolicyGenerator definition:

```
apiVersion: policyGenerator/v1
kind: PolicyGenerator
metadata:
  name: acm-policy
  namespace: acm-policy-generator
  # The arguments should be given and defined as below with same order --
  policyGenTempPath= --sourcePath= --outPath= --stdout --customResources
  argsOneLiner: ./ranPolicyGenTempExamples ./sourcePolicies ./out true false
```

Where:

- **policyGenTempPath** is the path to the **policyGenTemp** files.
- **sourcePath**: is the path to the source policies.
- **outPath**: is the path to save the generated ACM policies.
- **stdout**: If **true**, prints the generated policies to the console.
- **customResources**: If **true** generates the CRs from the **sourcePolicies** files without ACM policies.

2. Test PolicyGen by running the following commands:

```
$ cd cnf-features-deploy/ztp/ztp-policy-generator/
```

```
$ XDG_CONFIG_HOME=./ kustomize build --enable-alpha-plugins
```

An **out** directory is created with the expected policies, as shown in this example:

```
out
├── common
│   ├── common-log-sub-ns-policy.yaml
│   ├── common-log-sub-oper-policy.yaml
│   ├── common-log-sub-policy.yaml
│   ├── common-nto-sub-catalog-policy.yaml
│   ├── common-nto-sub-ns-policy.yaml
│   ├── common-nto-sub-oper-policy.yaml
│   ├── common-nto-sub-policy.yaml
│   ├── common-policies-placementbinding.yaml
│   ├── common-policies-placementrule.yaml
│   ├── common-ptp-sub-ns-policy.yaml
│   ├── common-ptp-sub-oper-policy.yaml
│   ├── common-ptp-sub-policy.yaml
│   ├── common-sriov-sub-ns-policy.yaml
│   ├── common-sriov-sub-oper-policy.yaml
│   └── common-sriov-sub-policy.yaml
├── groups
│   ├── group-du
│   │   ├── group-du-mc-mount-ns-policy.yaml
│   │   ├── group-du-mcp-du-policy.yaml
│   │   ├── group-du-mc-sctp-policy.yaml
│   │   ├── group-du-policies-placementbinding.yaml
│   │   ├── group-du-policies-placementrule.yaml
│   │   ├── group-du-ptp-config-policy.yaml
│   │   └── group-du-sriov-operconfig-policy.yaml
│   ├── group-sno-du
│   │   ├── group-du-sno-policies-placementbinding.yaml
│   │   ├── group-du-sno-policies-placementrule.yaml
│   │   ├── group-sno-du-console-policy.yaml
│   │   ├── group-sno-du-log-forwarder-policy.yaml
│   │   └── group-sno-du-log-policy.yaml
└── sites
    └── site-du-sno-1
        ├── site-du-sno-1-policies-placementbinding.yaml
        └── site-du-sno-1-policies-placementrule.yaml
```

```

|— site-du-sno-1-sriov-nn-fh-policy.yaml
|— site-du-sno-1-sriov-nnp-mh-policy.yaml
|— site-du-sno-1-sriov-nw-fh-policy.yaml
|— site-du-sno-1-sriov-nw-mh-policy.yaml
|— site-du-sno-1-.yaml

```

The common policies are flat because they will be applied to all clusters. However, the groups and sites have subdirectories for each group and site as they will be applied to different clusters.

23.28.1. Troubleshooting the managed cluster

Use this procedure to diagnose any installation issues that might occur with the managed clusters.

Procedure

1. Check the status of the managed cluster:

```
$ oc get managedcluster
```

Example output

NAME	HUB ACCEPTED	MANAGED CLUSTER	URLS	JOINED	AVAILABLE
AGE					
SNO-cluster	true	True	True	2d19h	

If the status in the **AVAILABLE** column is **True**, the managed cluster is being managed by the hub.

If the status in the **AVAILABLE** column is **Unknown**, the managed cluster is not being managed by the hub. Use the following steps to continue checking to get more information.

2. Check the **AgentClusterInstall** install status:

```
$ oc get clusterdeployment -n <cluster_name>
```

Example output

NAME	PLATFORM	REGION	CLUSTERTYPE	INSTALLED	INFRAID
VERSION	POWERSTATE	AGE			
Sno0026	agent-baremetal		false	Initialized	
2d14h					

If the status in the **INSTALLED** column is **false**, the installation was unsuccessful.

3. If the installation failed, enter the following command to review the status of the **AgentClusterInstall** resource:

```
$ oc describe agentclusterinstall -n <cluster_name> <cluster_name>
```

4. Resolve the errors and reset the cluster:
 - a. Remove the cluster's managed cluster resource:

```
$ oc delete managedcluster <cluster_name>
```

- b. Remove the cluster's namespace:

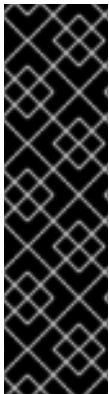
```
$ oc delete namespace <cluster_name>
```

This deletes all of the namespace-scoped custom resources created for this cluster. You must wait for the **ManagedCluster** CR deletion to complete before proceeding.

- c. Recreate the custom resources for the managed cluster.

23.29. UPDATING MANAGED POLICIES WITH THE TOPOLOGY AWARE LIFECYCLE MANAGER

You can use the Topology Aware Lifecycle Manager (TALM) to manage the software lifecycle of multiple OpenShift clusters. TALM uses Red Hat Advanced Cluster Management (RHACM) policies to perform changes on the target clusters.



IMPORTANT

The Topology Aware Lifecycle Manager is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see <https://access.redhat.com/support/offerings/techpreview/>.

Additional resources

- For more information about the Topology Aware Lifecycle Manager, see [About the Topology Aware Lifecycle Manager](#).

23.29.1. About the auto-created ClusterGroupUpgrade CR for ZTP

TALM has a controller called **ManagedClusterForCGU** that monitors the **Ready** state of the **ManagedCluster** CRs on the hub cluster and creates the **ClusterGroupUpgrade** CRs for ZTP (zero touch provisioning).

For any managed cluster in the **Ready** state without a "ztp-done" label applied, the **ManagedClusterForCGU** controller automatically creates a **ClusterGroupUpgrade** CR in the **ztp-install** namespace with its associated RHACM policies that are created during the ZTP process. TALM then remediates the set of configuration policies that are listed in the auto-created **ClusterGroupUpgrade** CR to push the configuration CRs to the managed cluster.



NOTE

If the managed cluster has no bound policies when the cluster becomes **Ready**, no **ClusterGroupUpgrade** CR is created.

Example of an auto-created ClusterGroupUpgrade CR for ZTP

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  generation: 1
  name: spoke1
  namespace: ztp-install
  ownerReferences:
  - apiVersion: cluster.open-cluster-management.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: ManagedCluster
    name: spoke1
    uid: 98fdb9b2-51ee-4ee7-8f57-a84f7f35b9d5
  resourceVersion: "46666836"
  uid: b8be9cd2-764f-4a62-87d6-6b767852c7da
spec:
  actions:
    afterCompletion:
      addClusterLabels:
        ztp-done: "" ❶
      deleteClusterLabels:
        ztp-running: ""
      deleteObjects: true
    beforeEnable:
      addClusterLabels:
        ztp-running: "" ❷
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - common-spoke1-config-policy
  - common-spoke1-subscriptions-policy
  - group-spoke1-config-policy
  - spoke1-config-policy
  - group-spoke1-validator-du-policy
  preCaching: false
  remediationStrategy:
    maxConcurrency: 1
    timeout: 240

```

- ❶ Applied to the managed cluster when TALM completes the cluster configuration.
- ❷ Applied to the managed cluster when TALM starts deploying the configuration policies.

23.30. END-TO-END PROCEDURES FOR UPDATING CLUSTERS IN A DISCONNECTED ENVIRONMENT

If you have deployed spoke clusters with distributed unit (DU) profiles using the GitOps ZTP with the Topology Aware Lifecycle Manager (TALM) pipeline described in "Deploying distributed units at scale in a disconnected environment", this procedure describes how to upgrade your spoke clusters and Operators.

23.30.1. Preparing for the updates

This procedure makes use of the Topology Aware Lifecycle Manager (TALM) which requires the 4.10 version or later of the ZTP container for compatibility.

23.30.2. Setting up the environment

TALM can perform both platform and Operator updates.

You must mirror both the platform image and Operator images that you want to update to in your mirror registry before you can use TALM to update your disconnected clusters. Complete the following steps to mirror the images:

- For platform updates, you must perform the following steps:
 1. Mirror the desired OpenShift Container Platform image repository. Ensure that the desired platform image is mirrored by following the "Mirroring the OpenShift Container Platform image repository" procedure linked in the Additional Resources. Save the contents of the **imageContentSources** section in the **imageContentSources.yaml** file:

Example output

```
imageContentSources:
- mirrors:
  - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-dev/openshift4
    source: quay.io/openshift-release-dev/ocp-release
- mirrors:
  - mirror-ocp-registry.ibmcloud.io.cpak:5000/openshift-release-dev/openshift4
    source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

2. Save the image signature of the desired platform image that was mirrored. You must add the image signature to the **PolicyGenTemplate** CR for platform updates. To get the image signature, perform the following steps:
 - a. Specify the desired OpenShift Container Platform tag by running the following command:

```
$ OCP_RELEASE_NUMBER=<release_version>
```

- b. Specify the architecture of the server by running the following command:

```
$ ARCHITECTURE=<server_architecture>
```

- c. Get the release image digest from Quay by running the following command

```
$ DIGEST="$(oc adm release info quay.io/openshift-release-dev/ocp-release:${OCP_RELEASE_NUMBER}-${ARCHITECTURE} | sed -n 's/Pull From: .*@//p')"
```

- d. Set the digest algorithm by running the following command:

```
$ DIGEST_ALGO="${DIGEST%%:*}"
```

- e. Set the digest signature by running the following command:

```
$ DIGEST_ENCODED="${DIGEST#*:}"
```


- f. Get the image signature from the mirror.openshift.com website by running the following command:

```
$ SIGNATURE_BASE64=$(curl -s "https://mirror.openshift.com/pub/openshift-
v4/signatures/openshift/release/${DIGEST_ALGO}=${DIGEST_ENCODED}/signature
-1" | base64 -w0 && echo)
```

- g. Save the image signature to the **checksum-<OCP_RELEASE_NUMBER>.yaml** file by running the following commands:

```
$ cat >checksum-${OCP_RELEASE_NUMBER}.yaml <<EOF
${DIGEST_ALGO}-${DIGEST_ENCODED}: ${SIGNATURE_BASE64}
EOF
```

3. Prepare the update graph. You have two options to prepare the update graph:

- a. Use the OpenShift Update Service.
For more information about how to set up the graph on the hub cluster, see [Deploy the operator for OpenShift Update Service](#) and [Build the graph data init container](#).
- b. Make a local copy of the upstream graph. Host the update graph on an **http** or **https** server in the disconnected environment that has access to the spoke cluster. To download the update graph, use the following command:

```
$ curl -s https://api.openshift.com/api/upgrades_info/v1/graph?channel=stable-4.11 -
o ~/upgrade-graph_stable-4.11
```

- For Operator updates, you must perform the following task:
 - Mirror the Operator catalogs. Ensure that the desired operator images are mirrored by following the procedure in the "Mirroring Operator catalogs for use with disconnected clusters" section.

Additional resources

- For more information about how to update ZTP, see [Upgrading GitOps ZTP](#).
- For more information about how to mirror an OpenShift Container Platform image repository, see [Mirroring the OpenShift Container Platform image repository](#).
- For more information about how to mirror Operator catalogs for disconnected clusters, see [Mirroring Operator catalogs for use with disconnected clusters](#).
- For more information about how to prepare the disconnected environment and mirroring the desired image repository, see [Preparing the disconnected environment](#).
- For more information about update channels and releases, see [Understanding upgrade channels and releases](#).

23.30.3. Performing a platform update

You can perform a platform update with the TALM.

Prerequisites

- Install the Topology Aware Lifecycle Manager (TALM).
- Update ZTP to the latest version.
- Provision one or more managed clusters with ZTP.
- Mirror the desired image repository.
- Log in as a user with **cluster-admin** privileges.
- Create RHACM policies in the hub cluster.

Procedure

1. Create a **PolicyGenTemplate** CR for the platform update:
 - a. Save the following contents of the **PolicyGenTemplate** CR in the **du-upgrade.yaml** file.

Example of PolicyGenTemplate for platform update

```
apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "du-upgrade"
  namespace: "ztp-group-du-sno"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  remediationAction: inform
  sourceFiles:
    - fileName: ImageSignature.yaml 1
      policyName: "platform-upgrade-prep"
      binaryData:
        ${DIGEST_ALGO}-${DIGEST_ENCODED}: ${SIGNATURE_BASE64} 2
    - fileName: DisconnectedICSP.yaml
      policyName: "platform-upgrade-prep"
      metadata:
        name: disconnected-internal-icsp-for-ocp
      spec:
        repositoryDigestMirrors: 3
        - mirrors:
            - quay-intern.example.com/ocp4/openshift-release-dev
              source: quay.io/openshift-release-dev/ocp-release
          - mirrors:
            - quay-intern.example.com/ocp4/openshift-release-dev
              source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
    - fileName: ClusterVersion.yaml 4
      policyName: "platform-upgrade-prep"
      metadata:
        name: version
        annotations:
          ran.openshift.io/ztp-deploy-wave: "1"
      spec:
        channel: "stable-4.11"
        upstream: http://upgrade.example.com/images/upgrade-graph_stable-4.11
```

```
- fileName: ClusterVersion.yaml 5
  policyName: "platform-upgrade"
  metadata:
    name: version
  spec:
    channel: "stable-4.11"
    upstream: http://upgrade.example.com/images/upgrade-graph_stable-4.11
    desiredUpdate:
      version: 4.11.4
  status:
    history:
      - version: 4.11.4
      state: "Completed"
```

- 1** The **ConfigMap** CR contains the signature of the desired release image to update to.
- 2** Shows the image signature of the desired OpenShift Container Platform release. Get the signature from the **checksum-\${OCP_RELEASE_NUMBER}.yaml** file you saved when following the procedures in the "Setting up the environment" section.
- 3** Shows the mirror repository that contains the desired OpenShift Container Platform image. Get the mirrors from the **imageContentSources.yaml** file that you saved when following the procedures in the "Setting up the environment" section.
- 4** Shows the **ClusterVersion** CR to update upstream.
- 5** Shows the **ClusterVersion** CR to trigger the update. The **channel**, **upstream**, and **desiredVersion** fields are all required for image pre-caching.

The **PolicyGenTemplate** CR generates two policies:

- The **du-upgrade-platform-upgrade-prep** policy does the preparation work for the platform update. It creates the **ConfigMap** CR for the desired release image signature, creates the image content source of the mirrored release image repository, and updates the cluster version with the desired update channel and the update graph reachable by the spoke cluster in the disconnected environment.
 - The **du-upgrade-platform-upgrade** policy is used to perform platform upgrade.
- b. Add the **du-upgrade.yaml** file contents to the **kustomization.yaml** file located in the ZTP Git repository for the **PolicyGenTemplate** CRs and push the changes to the Git repository. ArgoCD pulls the changes from the Git repository and generates the policies on the hub cluster.
 - c. Check the created policies by running the following command:

```
$ oc get policies -A | grep platform-upgrade
```

2. Apply the required update resources before starting the platform update with the TALM.
 - a. Save the content of the **platform-upgrade-prep ClusterUpgradeGroup** CR with the **du-upgrade-platform-upgrade-prep** policy and the target spoke clusters to the **cgu-platform-upgrade-prep.yml** file, as shown in the following example:

```
apiVersion: ran.openshift.io/v1alpha1
```

```
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-upgrade-prep
  namespace: default
spec:
  managedPolicies:
    - du-upgrade-platform-upgrade-prep
  clusters:
    - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: true
```

- b. Apply the policy to the hub cluster by running the following command:

```
$ oc apply -f cgu-platform-upgrade-prep.yml
```

- c. Monitor the update process. Upon completion, ensure that the policy is compliant by running the following command:

```
$ oc get policies --all-namespaces
```

3. Create the **ClusterGroupUpdate** CR for the platform update with the **spec.enable** field set to **false**.

- a. Save the content of the platform update **ClusterGroupUpdate** CR with the **du-upgrade-platform-upgrade** policy and the target clusters to the **cgu-platform-upgrade.yml** file, as shown in the following example:

```
apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-platform-upgrade
  namespace: default
spec:
  managedPolicies:
    - du-upgrade-platform-upgrade
  preCaching: false
  clusters:
    - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false
```

- b. Apply the **ClusterGroupUpdate** CR to the hub cluster by running the following command:

```
$ oc apply -f cgu-platform-upgrade.yml
```

4. Optional: Pre-cache the images for the platform update.

- a. Enable pre-caching in the **ClusterGroupUpdate** CR by running the following command:

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-platform-
upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

- b. Monitor the update process and wait for the pre-caching to complete. Check the status of pre-caching by running the following command on the hub cluster:

```
$ oc get cgu cgu-platform-upgrade -o jsonpath='{.status.precaching.status}'
```

5. Start the platform update:

- a. Enable the **cgu-platform-upgrade** policy and disable pre-caching by running the following command:

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-platform-
upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

- b. Monitor the process. Upon completion, ensure that the policy is compliant by running the following command:

```
$ oc get policies --all-namespaces
```

Additional resources

- For more information about mirroring the images in a disconnected environment, [Preparing the disconnected environment](#)

23.30.4. Performing an Operator update

You can perform an Operator update with the TALM.

Prerequisites

- Install the Topology Aware Lifecycle Manager (TALM).
- Update ZTP to the latest version.
- Provision one or more managed clusters with ZTP.
- Mirror the desired index image, bundle images, and all Operator images referenced in the bundle images.
- Log in as a user with **cluster-admin** privileges.
- Create RHACM policies in the hub cluster.

Procedure

1. Update the **PolicyGenTemplate** CR for the Operator update.
 - a. Update the **du-upgrade PolicyGenTemplate** CR with the following additional contents in the **du-upgrade.yaml** file:

```

apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "du-upgrade"
  namespace: "ztp-group-du-sno"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  remediationAction: inform
  sourceFiles:
    - fileName: DefaultCatsrc.yaml
      remediationAction: inform
      policyName: "operator-catsrc-policy"
      metadata:
        name: redhat-operators
      spec:
        displayName: Red Hat Operators Catalog
        image: registry.example.com:5000/olm/redhat-operators:v4.11 1
        updateStrategy: 2
        registryPoll:
          interval: 1h

```

- 1** The index image URL contains the desired Operator images. If the index images are always pushed to the same image name and tag, this change is not needed.
- 2** Set how frequently the Operator Lifecycle Manager (OLM) polls the index image for new Operator versions with the **registryPoll.interval** field. This change is not needed if a new index image tag is always pushed for y-stream and z-stream Operator updates. The **registryPoll.interval** field can be set to a shorter interval to expedite the update, however shorter intervals increase computational load. To counteract this, you can restore **registryPoll.interval** to the default value once the update is complete.

- b. This update generates one policy, **du-upgrade-operator-catsrc-policy**, to update the **redhat-operators** catalog source with the new index images that contain the desired Operators images.



NOTE

If you want to use the image pre-caching for Operators and there are Operators from a different catalog source other than **redhat-operators**, you must perform the following tasks:

- Prepare a separate catalog source policy with the new index image or registry poll interval update for the different catalog source.
- Prepare a separate subscription policy for the desired Operators that are from the different catalog source.

For example, the desired SRIOV-FEC Operator is available in the **certified-operators** catalog source. To update the catalog source and the Operator subscription, add the following contents to generate two policies, **du-upgrade-fec-catsrc-policy** and **du-upgrade-subscriptions-fec-policy**:

```

apiVersion: ran.openshift.io/v1
kind: PolicyGenTemplate
metadata:
  name: "du-upgrade"
  namespace: "ztp-group-du-sno"
spec:
  bindingRules:
    group-du-sno: ""
  mcp: "master"
  remediationAction: inform
  sourceFiles:
    ...
    - fileName: DefaultCatsrc.yaml
      remediationAction: inform
      policyName: "fec-catsrc-policy"
      metadata:
        name: certified-operators
      spec:
        displayName: Intel SRIOV-FEC Operator
        image: registry.example.com:5000/olm/far-edge-sriov-fec:v4.10
        updateStrategy:
          registryPoll:
            interval: 10m
    - fileName: AcceleratorsSubscription.yaml
      policyName: "subscriptions-fec-policy"
      spec:
        channel: "stable"
        source: certified-operators

```

- c. Remove the specified subscriptions channels in the common **PolicyGenTemplate** CR, if they exist. The default subscriptions channels from the ZTP image are used for the update.



NOTE

The default channel for the Operators applied through ZTP 4.11 is **stable**, except for the **performance-addon-operator**. As of OpenShift Container Platform 4.11, the **performance-addon-operator** functionality was moved to the **node-tuning-operator**. For the 4.10 release, the default channel for PAO is **v4.10**. You can also specify the default channels in the common **PolicyGenTemplate** CR.

- d. Push the **PolicyGenTemplate** CRs updates to the ZTP Git repository. ArgoCD pulls the changes from the Git repository and generates the policies on the hub cluster.
- e. Check the created policies by running the following command:

```
$ oc get policies -A | grep -E "catsrc-policy|subscription"
```

2. Apply the required catalog source updates before starting the Operator update.
 - a. Save the content of the **ClusterGroupUpgrade** CR named **operator-upgrade-prep** with the catalog source policies and the target spoke clusters to the **cgu-operator-upgrade-prep.yml** file:

■

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-operator-upgrade-prep
  namespace: default
spec:
  clusters:
  - spoke1
  enable: true
  managedPolicies:
  - du-upgrade-operator-catsrc-policy
  remediationStrategy:
    maxConcurrency: 1

```

- b. Apply the policy to the hub cluster by running the following command:

```
$ oc apply -f cgu-operator-upgrade-prep.yml
```

- c. Monitor the update process. Upon completion, ensure that the policy is compliant by running the following command:

```
$ oc get policies -A | grep -E "catsrc-policy"
```

3. Create the **ClusterGroupUpgrade** CR for the Operator update with the **spec.enable** field set to **false**.

- a. Save the content of the Operator update **ClusterGroupUpgrade** CR with the **du-upgrade-operator-catsrc-policy** policy and the subscription policies created from the common **PolicyGenTemplate** and the target clusters to the **cgu-operator-upgrade.yml** file, as shown in the following example:

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-operator-upgrade
  namespace: default
spec:
  managedPolicies:
  - du-upgrade-operator-catsrc-policy ❶
  - common-subscriptions-policy ❷
  preCaching: false
  clusters:
  - spoke1
  remediationStrategy:
    maxConcurrency: 1
  enable: false

```

- ❶ The policy is needed by the image pre-caching feature to retrieve the operator images from the catalog source.
- ❷ The policy contains Operator subscriptions. If you have followed the structure and content of the reference **PolicyGenTemplates**, all Operator subscriptions are grouped into the **common-subscriptions-policy** policy.

**NOTE**

One **ClusterGroupUpgrade** CR can only pre-cache the images of the desired Operators defined in the subscription policy from one catalog source included in the **ClusterGroupUpgrade** CR. If the desired Operators are from different catalog sources, such as in the example of the SRIOV-FEC Operator, another **ClusterGroupUpgrade** CR must be created with **du-upgrade-fec-catsrc-policy** and **du-upgrade-subscriptions-fec-policy** policies for the SRIOV-FEC Operator images pre-caching and update.

- b. Apply the **ClusterGroupUpgrade** CR to the hub cluster by running the following command:

```
$ oc apply -f cgu-operator-upgrade.yml
```

4. Optional: Pre-cache the images for the Operator update.

- a. Before starting image pre-caching, verify the subscription policy is **NonCompliant** at this point by running the following command:

```
$ oc get policy common-subscriptions-policy -n <policy_namespace>
```

Example output

```
NAME                                REMEDIATION ACTION  COMPLIANCE STATE  AGE
common-subscriptions-policy  inform              NonCompliant      27d
```

- b. Enable pre-caching in the **ClusterGroupUpgrade** CR by running the following command:

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-operator-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

- c. Monitor the process and wait for the pre-caching to complete. Check the status of pre-caching by running the following command on the spoke cluster:

```
$ oc get cgu cgu-operator-upgrade -o jsonpath='{.status.precaching.status}'
```

- d. Check if the pre-caching is completed before starting the update by running the following command:

```
$ oc get cgu -n default cgu-operator-upgrade -ojsonpath='{.status.conditions}' | jq
```

Example output

```
[
  {
    "lastTransitionTime": "2022-03-08T20:49:08.000Z",
    "message": "The ClusterGroupUpgrade CR is not enabled",
    "reason": "UpgradeNotStarted",
    "status": "False",
    "type": "Ready"
  },
  {
```

```

    "lastTransitionTime": "2022-03-08T20:55:30.000Z",
    "message": "Precaching is completed",
    "reason": "PrecachingCompleted",
    "status": "True",
    "type": "PrecachingDone"
  }
]

```

5. Start the Operator update.

- a. Enable the **cgu-operator-upgrade ClusterGroupUpgrade** CR and disable pre-caching to start the Operator update by running the following command:

```

$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-operator-
upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge

```

- b. Monitor the process. Upon completion, ensure that the policy is compliant by running the following command:

```

$ oc get policies --all-namespaces

```

Additional resources

- For more information about updating GitOps ZTP, see [Upgrading GitOps ZTP](#).

23.30.5. Performing a platform and an Operator update together

You can perform a platform and an Operator update at the same time.

Prerequisites

- Install the Topology Aware Lifecycle Manager (TALM).
- Update ZTP to the latest version.
- Provision one or more managed clusters with ZTP.
- Log in as a user with **cluster-admin** privileges.
- Create RHACM policies in the hub cluster.

Procedure

1. Create the **PolicyGenTemplate** CR for the updates by following the steps described in the "Performing a platform update" and "Performing an Operator update" sections.
2. Apply the prep work for the platform and the Operator update.
 - a. Save the content of the **ClusterGroupUpgrade** CR with the policies for platform update preparation work, catalog source updates, and target clusters to the **cgu-platform-operator-upgrade-prep.yml** file, for example:

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade

```

```

metadata:
  name: cgu-platform-operator-upgrade-prep
  namespace: default
spec:
  managedPolicies:
    - du-upgrade-platform-upgrade-prep
    - du-upgrade-operator-catsrc-policy
  clusterSelector:
    - group-du-sno
  remediationStrategy:
    maxConcurrency: 10
  enable: true

```

- b. Apply the **cgu-platform-operator-upgrade-prep.yml** file to the hub cluster by running the following command:

```
$ oc apply -f cgu-platform-operator-upgrade-prep.yml
```

- c. Monitor the process. Upon completion, ensure that the policy is compliant by running the following command:

```
$ oc get policies --all-namespaces
```

3. Create the **ClusterGroupUpdate** CR for the platform and the Operator update with the **spec.enable** field set to **false**.

- a. Save the contents of the platform and Operator update **ClusterGroupUpdate** CR with the policies and the target clusters to the **cgu-platform-operator-upgrade.yml** file, as shown in the following example:

```

apiVersion: ran.openshift.io/v1alpha1
kind: ClusterGroupUpgrade
metadata:
  name: cgu-du-upgrade
  namespace: default
spec:
  managedPolicies:
    - du-upgrade-platform-upgrade ❶
    - du-upgrade-operator-catsrc-policy ❷
    - common-subscriptions-policy ❸
  preCaching: true
  clusterSelector:
    - group-du-sno
  remediationStrategy:
    maxConcurrency: 1
  enable: false

```

- ❶ This is the platform update policy.
- ❷ This is the policy containing the catalog source information for the Operators to be updated. It is needed for the pre-caching feature to determine which Operator images to download to the spoke cluster.
- ❸ This is the policy to update the Operators.

- b. Apply the **cgu-platform-operator-upgrade.yml** file to the hub cluster by running the following command:

```
$ oc apply -f cgu-platform-operator-upgrade.yml
```

4. Optional: Pre-cache the images for the platform and the Operator update.

- a. Enable pre-caching in the **ClusterGroupUpgrade** CR by running the following command:

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-du-upgrade \
--patch '{"spec":{"preCaching": true}}' --type=merge
```

- b. Monitor the update process and wait for the pre-caching to complete. Check the status of pre-caching by running the following command on the spoke cluster:

```
$ oc get jobs,pods -n openshift-talm-pre-cache
```

- c. Check if the pre-caching is completed before starting the update by running the following command:

```
$ oc get cgu cgu-du-upgrade -ojsonpath='{.status.conditions}'
```

5. Start the platform and Operator update.

- a. Enable the **cgu-du-upgrade ClusterGroupUpgrade** CR to start the platform and the Operator update by running the following command:

```
$ oc --namespace=default patch clustergroupupgrade.ran.openshift.io/cgu-du-upgrade \
--patch '{"spec":{"enable":true, "preCaching": false}}' --type=merge
```

- b. Monitor the process. Upon completion, ensure that the policy is compliant by running the following command:

```
$ oc get policies --all-namespaces
```



NOTE

The CRs for the platform and Operator updates can be created from the beginning by configuring the setting to **spec.enable: true**. In this case, the update starts immediately after pre-caching completes and there is no need to manually enable the CR.

Both pre-caching and the update create extra resources, such as policies, placement bindings, placement rules, managed cluster actions, and managed cluster view, to help complete the procedures. Setting the **afterCompletion.deleteObjects** field to **true** deletes all these resources after the updates complete.

23.31. REMOVING PERFORMANCE ADDON OPERATOR SUBSCRIPTIONS FROM DEPLOYED CLUSTERS

In earlier versions of OpenShift Container Platform, the Performance Addon Operator provided automatic, low latency performance tuning for applications. In OpenShift Container Platform 4.11 or later, these functions are part of the Node Tuning Operator.

Do not install the Performance Addon Operator on clusters running OpenShift Container Platform 4.11 or later. If you upgrade to OpenShift Container Platform 4.11 or later, the Node Tuning Operator automatically removes the Performance Addon Operator. However, you need to manually remove any policies that create Performance Addon Operator subscriptions to prevent a reinstallation of the Operator. The reference DU profile includes the Performance Addon Operator in the **common-ranGen.yaml PolicyGenTemplate**. To remove the subscription from deployed spoke clusters, you must update **common-ranGen.yaml**.



NOTE

If you install Performance Addon Operator 4.10.3-5 or later on OpenShift Container Platform 4.11 or later, the Performance Addon Operator detects the cluster version and automatically hibernates to avoid interfering with the Node Tuning Operator functions. However, to ensure best performance, remove the Performance Addon Operator from your OpenShift Container Platform 4.11 clusters.

Prerequisites

- Create a Git repository where you manage your custom site configuration data. The repository must be accessible from the hub cluster and be defined as a source repository for Argo CD.
- Update to OpenShift Container Platform 4.11 or later.
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Change the **complianceType** to **mustnothave** for the Performance Addon Operator namespace, Operator group, and subscription in the **common-ranGen.yaml** file.

```
- fileName: PaoSubscriptionNS.yaml
  policyName: "subscriptions-policy"
  complianceType: mustnothave
- fileName: PaoSubscriptionOperGroup.yaml
  policyName: "subscriptions-policy"
  complianceType: mustnothave
- fileName: PaoSubscription.yaml
  policyName: "subscriptions-policy"
  complianceType: mustnothave
```

2. Merge the changes with your custom site repository and wait for the ArgoCD application to synchronize the change to the hub cluster. The status of the **common-subscriptions-policy** policy changes to **Non-Compliant**.
3. Apply the change to your target clusters by using the Topology Aware Lifecycle Manager. For more information about rolling out configuration changes, see the *Additional resources* section.
4. Monitor the process. When the status of the **common-subscriptions-policy** policy for a target cluster is **Compliant**, the Performance Addon Operator has been removed from the cluster. Get the status of the **common-subscriptions-policy** by running the following command:

```
$ oc get policy -n common-subscriptions-policy
```

5. Delete the Performance Addon Operator namespace, Operator group and subscription CRs from **.spec.sourceFiles** in the **common-ranGen.yaml** file.
6. Merge the changes with your custom site repository and wait for the ArgoCD application to synchronize the change to the hub cluster. The policy remains compliant.

Additional resources

- [Upgrading GitOps ZTP](#)

CHAPTER 24. REQUESTING CRI-O AND KUBELET PROFILING DATA BY USING THE NODE OBSERVABILITY OPERATOR

The Node Observability Operator collects and stores the CRI-O and Kubelet profiling data of worker nodes. You can query the profiling data to analyze the CRI-O and Kubelet performance trends and debug the performance related issues.



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 <https://access.redhat.com/support/offerings/techpreview/>.

24.1. 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.

24.2. 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.

24.2.1. 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:
    - node-observability-operator
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
```

24.2.2. 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

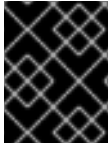
1. Log in to the OpenShift Container Platform web console.
2. In the Administrator's navigation panel, expand **Operators** → **OperatorHub**.
3. In the **All items** field, enter **Node Observability Operator** and select the **Node Observability Operator** tile.
4. Click **Install**.
5. On the **Install 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.

24.3. 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.



IMPORTANT

Creating a **NodeObservability** CR reboots all the worker nodes. It 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/v1alpha1
kind: NodeObservability
metadata:
  name: cluster 1
spec:
  labels:
    node-role.kubernetes.io/worker: ""
  type: crio-kubelet
```

1

You must specify the name as **cluster** because there should be only one **NodeObservability** CR per cluster.

- Run the **NodeObservability** CR:

```
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'
```

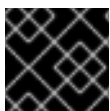
Example output

```
conditions:
conditions:
- lastTransitionTime: "2022-07-05T07:33:54Z"
  message: 'DaemonSet node-observability-ds ready: true NodeObservabilityMachineConfig
  ready: true'
  reason: Ready
  status: "True"
  type: Ready
```

NodeObservability CR run is completed when the reason is **Ready** and the status is **True**.

24.4. 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.



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

- Create a **NodeObservabilityRun** resource file named **nodeobservabilityrun.yaml** that contains the following text:

```
apiVersion: nodeobservability.olm.openshift.io/v1alpha1
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 -o yaml | yq '.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 | yq
.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
```